special - ASSEMBLER - special



Nr. 3/86 DM 19,80 / ÖS 158 / SFR 19,80

Über 90 Seiten Alles über Assembler für den TI 99/4A

ASSEMBLER

ASSEMBLER

MACHEN SIE MEHR AUS IHREM TI MIT ASSEMBLER

Aus dem Inhalt:

- DELETE
- BILDSCHIRM SPEICHERN
- SCROLL IN AUSSCHNITTEN
- HILFE FÜR DAS MINI MEM
- FASTCOPY
- BIT MAP MODE
- ASSEMBLER IN TI BASIC
- ZEICHENDEFINITION

ASSEMBLER

Ein Muß für jeden Assembler-Anwender!

ASSEMBLER

special - ASSEMBLER - special



REVUE

IMPRESSUM

TI ASSEMBLER ist eine Sonderpublikation der TI REVUE in der AKTUELL GRUPPE Werner E. Seibt.
Verantwortlich für den Inhalt: Heiner Martin
Verantwortlich für Anzeigen: Bruno G. Redase.
Alle: Postfach 1107 in 8044 Lohhof.
Anfragen bitte nur schriftlich.
Druck: Maier und Söhne
Es gilt die Honorarliste des Verlages.
Für unaufgefordert eingesandte Manuskripte und Listings keine Haftung
Bei Einsendung von Texten, Fotos und Programmträgern erteilt der Autor dem Verlag die Genehmigung für einen einmaligen Abdruck sowie die Aufnahme in den Programm-Service nach den Verlags-Sätzen!

Namentlich gezeichnete Artikel geben nicht unbedingt die Meinung der Redaktion wieder.
Kein Anspruch auf Lieferung bei Ausfall durch höhere Gewalt.
Gerichtsstand: München
Geschäftsführer: Werner E. Seibt
Abo- und Kassetten-Service: Henny Rose Seibt

**Wie immer steht unseren Lesern unser Telefon-Service zur Verfügung! Jeden Dienstag von 15 bis 19 Uhr.
Für technische Fragen: 0731/33220 und
für Listings/Programme: 089/1298013**

special - ASSEMBLER - special



SONDERHEFT
Nr. 3/86/März

INHALT

Grüß Gott - Gruezi - Guten Tag

Mit diesem Sonderheft halten Sie das erste der TI-REVUE in der Hand, welches sich ausschließlich mit Assemblerprogrammen beschäftigt. Ganz auf Basicprogramme konnten wir allerdings auch hier nicht verzichten, ein sehr komfortabler Basic-Assembler für das Mini-Memory ist in diesem Sonderheft auch enthalten. Für die Besitzer des Extended Basic und einer Speichererweiterung sind bei diesem Assembler alle notwendigen Änderungen angegeben, so daß sich das Programm auch damit einsetzen läßt. Es ist dann allerdings noch nicht durch Mehrfachanweisungen in jeder Zeile für das Extended Basic optimiert. Dies dürfte aber wohl einfach zu realisieren sein. Damit kann dann noch ein bißchen Geschwindigkeit herausgeholt werden.
Bei der Zusammenstellung der Programme haben wir uns bemüht, ein möglichst breites Spektrum verschiedener Anwendungen zu treffen. Auch an notwendige Utilities haben wir gedacht. Viele der Programme geben über den eigentlichen Anwendungszweck hinaus wertvolle Tips zur Assemblerprogrammierung. So sind zwei Utilities, die beim Extended Basic fehlen, DSRLNK und GPLLNK, in diesem Sonderheft in sehr kurzen Ausführungen enthalten. Anhand eines Sektorkopierprogramms wird der Zugriff auf einzelne Sektoren von Disketten gezeigt. Eine besondere Art der Speicherplatzreservierung im VDP-RAM unter Extended Basic ist im Programm interruptgesteuerte Soundlistenverarbeitung enthalten. Diese Reservierung arbeitet übrigens nach den bisherigen Erkenntnissen auch unter TI-Basic. Das waren nur einzelne Beispiele und wir glauben, daß Ihnen die Programme jede Menge Anregungen geben können.
Zum Schluß sei noch ein Wort zum Rücksprung aus dem Assembler-Programm in das Basic gesagt. Texas Instruments schreibt hier vor, daß das Condition-Bit im GPL-Statusregister gelöscht wird und dann eine Verzweigung zur Adresse >0070 geschieht. Nun sind viele Anwender darauf gekommen, daß dies einfacher mit einer direkten Verzweigung zur Adresse >006A geschehen kann. Dies muß, da diese Adresse nicht unbedingt garantiert ist, d.h. es kann einzelne TI 99/4A-Konsolen geben, bei denen dies nicht richtig arbeitet, eventuell jeweils geändert werden.
Wir hoffen, daß wir mit der Auswahl der Programme Ihren Geschmack getroffen haben und wünschen Ihnen eine weitere erfolgreiche Arbeit in Assembler.
Die Redaktion

SERVICE

LISTINGS

ASSEMBLER LEICHT GEMACHT FÜR DEN TI 99/4A

Auf den nächsten Seiten wollen wir ein bißchen näher auf das Thema Maschinensprache mit dem TI 99/4A eingehen. Dabei sollen nicht nur das Editor/Assembler Modul, sondern auch die Module Extended Basic und Mini-Memory angesprochen werden. Beginnen wollen wir ganz einfach, aber am Schluß jedes Beitrages sollen auch für die Könner einige spezielle Tricks verraten werden.

Was brauchen wir, um mit Assembler arbeiten zu können? Auf jeden Fall eines der eben schon angesprochenen Module, wobei hier das neue Extended Basic II plus zum Extended Basic zählt, auch wenn es eine besonders nützliche Routine zum Abspeichern von Assembler Programmen auf Kassette besitzt, ähnlich wie das Mini-Memory.

DAS MINI-MEMORY:

Mit diesem Modul kann man allein mit der Konsole zusammen schon in Maschinensprache (Assembler) programmieren. Eine eventuell vorhandene Speichererweiterung ist ebenfalls nutzbar. Ein Line-by-Line Assembler, also ein Programm, welches die Bildschirmeingaben in direkt vom 9900-Prozessor verwendbaren Code umwandelt, gehört zum Lieferumfang dieses Moduls. Zum Abspeichern der Assemblerroutinen genügt beim Mini-Memory ein Kassettenrekorder, ein Abspeichern auf Diskette ist nicht möglich, jedoch können mit dem Mini-Memory vom Editor/Assembler auf Diskette erstellte Maschinenprogramme verarbeitet werden.

DAS EXTENDED BASIC:

Bei diesem Modul ist unbedingt eine Speichererweiterung notwendig. Das Modul enthält auch kein Assembler-Programm, welches ebenfalls noch benötigt wird, allerdings wurden schon mehrere derartige Programme veröffentlicht. Zusammen mit einem Diskettenlaufwerk können vom Editor/Assembler erstellte Assemblerprogramme benutzt werden. Problematisch ist das Abspeichern der mit einem Extended Basic Assembler erstellten Routinen. Hier sind nochmals Hilfsprogramme notwendig.

DER EDITOR/ ASSEMBLER:

Er ist ohne Zweifel das komfortabelste Modul für die Assemblerprogrammierung, verfügt es doch über einen Editor, mit dem der Quellcode deutlich lesbar erstellt wird. Dieser Quellcode wird von einem Assembler, der auch einen kompletten Ausdruck auf einem Drucker erlaubt, in den Objektcode umgewandelt, welcher dann wieder zum Abarbeiten geladen wird. Allerdings erfordert dieses Modul auch die größte Ausrüstung: Speichererweiterung und Diskettenlaufwerk.

GRUNDSÄTZLICHES

Beim TI 99/4A wird Assembler nicht mit dem „Hammer programmiert", wie bei vielen anderen Computern dieser Klasse. Ursprünglich dachte Texas Instruments wohl nur an den Editor/Assembler, und hier ist das erstellte Maschinenprogramm so richtig komfortabel: Eine relative Adressierung ist möglich, d.h. der Benutzer muß sich nicht um den belegten Speicherplatz kümmern, sondern das besorgt das Programm, welches den Objektcode einfach in einen freien Speicherbereich lädt und alle notwendigen Adressen dabei umrechnet. Der Aufruf des Programms (LINK) erfolgt nun auch nicht über irgendeine Systemadresse, die man sich permanent merken muß, sondern über einen Namen, der bis zu 6 Stellen lang sein darf und der in einer gesonderten Tabelle zusammen mit der dazugehörigen Einsprungadresse abgelegt ist. Diese Benutzerfreundlichkeit bedarf aber nun einiger Übersicht bei direkter Assemblierung, wie es mit dem Line-by-Line Assembler des Mini-Memorys und mit fast allen Extended-Basic Assemblern geschieht.

DIE LITERATUR

Unabdingbar notwendig zum Programmieren des TI 99/4A in Assembler ist das Handbuch zum Editor/Assembler. Dies ist in Englisch geschrieben, es gibt davon aber auch eine zwar nicht als exzellent zu bezeichnende, aber doch hilfreiche Übersetzung ins Deutsche für diejenigen, die des Englischen nicht sehr mächtig sind. Eine weitere, sehr gute Lektüre zum Thema 9900 Assembler gibt es leider nicht mehr: Das 16 Bit Microprozessor Kursbuch von Texas Instruments. Ebenfalls sehr wichtig ist natürlich die Anleitung zum jeweils verwendeten Assembler (bei Extended Basic und Mini-Memory.

DAS ERSTE PROGRAMM

Aller Anfang ist schwer, besonders bei der Programmierung in Maschinensprache. Zwei der wesentlichen Schwierigkeiten mit Assembler wollen wir gleich zu Anfang behandeln: Der Einsprung auf das Programm und das Darstellen eines Zeichens auf dem Bildschirm. Folgendes Programm soll als Beispiel dienen:

```
       DEF   START
       REF   VSBW
MYWS   BSS   32
START  LWPI  MYWS
       LI    0,>0190
       LI    1,>4100
       BLWP  §VSBW
ENDLOS LIMI  2
       LIMI  0
       JMP   ENDLOS
       END
```

Dieses Programm für das Editor/Assembler Modul wird unter dem Namen START aufgerufen, das beinhaltet die erste Zeile mit DEF. REF bedeutet Reference und ist eine weitere bedienungsfreundliche Funktion: Beim Laden des Programms wird

hier die REF/DEF-Tabelle abgesucht, ob der Begriff enthalten ist und wenn ja, wird der entsprechende Wert eingesetzt.
Dann werden mit BSS 32 Bytes (16 Worte) für den Workspace (Arbeitsbereich) des TMS 9900 reserviert. Dieser Prozessor besitzt ja nicht die sonst viel verbreiteten Register in der CPU, sondern benutzt dafür einen Bereich im Ram, auf den der Workspacecounter zeigt. Ab dieser Adresse werden 16 Register (R0 bis R15) benutzt. Als 16-Bit Prozessor umfassen diese Register natürlich je 2 Bytes, deshalb der Wert 32 bzw. >0020.
Das Programm beginnt damit, daß der Workspacecounter geladen wird, und zwar mit der Anfangsadresse des eben reservierten Speicherbereiches. Nun laden wir die Register R0 mit der Adresse, in die wir in das VDP-Ram schreiben wollen und das höherwertige Byte von R1 mit dem Wert >41, was dem ASCII Wert des Buchstabens A entspricht. Mit BLWP wird nun ein Unterprogramm aufgerufen. Solche Unterprogramme stellen alle Module, mit denen in Assembler gearbeitet werden kann, zur Verfügung, VSBW ist dabei eine Kurzform für VDP Single Byte Write, was soviel bedeutet wie ein Byte in das VDP-Ram schreiben.
Danach wird mit LIMI 2 der Interrupt freigegeben und mit LIMI 0 wieder unterbunden. Dies dient dazu, daß wir mit der Quit-Taste, die ja von der Interrupt-Routine abgefragt wird, aus der Endlosschleife zum Titelbild kommen können. Die Endlosschleife ist mit dem unbedingten Sprungbefehl JMP (Jump) realisiert.
Kommen wir nun zu den verschiedenen Assembler. Mit dem Editor/Assembler erstellen wir mittels des Editors die Quellcode-Datei und assemblieren diese dann mit dem Assembler. Dabei sollten wir die Dateien schon vom Namen her deutlich kennzeichnen, z.B. mit DSK1TESTQ (für Quellcode) und DSK1TESTC (für Objektcode). Bei der Abfrage OPTIONS des Assemblers geben wir RC oder nur R ein. Wird das C gewählt, wird ein sogenannter compressed Code erstellt, dieser kann vom Extended-Basic-Modul nicht geladen werden. Danach wählen wir den Abschnitt LOAD AND RUN, tippen DSK1. TESTC ein und starten das Programm über den Namen START.
Nun haben wir aber einen einfachen Assembler in Extended-Basic geschrieben oder den des Mini-Memorys. Hier sind ein paar Einschränkungen zu machen: DEF, REF und BSS sind meist nicht vorhanden. Wir beginnen also an einer freien Speicherstelle, merken uns die Adresse (z.B. >7000 für das Mini-Memory Adresse (z.B. >2500 unter X-Basic oder >7000 für das Mini-Memory) und springen 32 Bytes höher, d.h. auf >2500 bzw. >7D20.
nun beginnen wir mit dem Programm:

```
LWPI   >2500 (>7D00)
LI     R0,>0190
LI     R1,>4100
BLWP   §>2020 (>6024)
LIMI   2
LIMI   0
JMP    >2530 (>7D30)
END
```

Die Werte in Klammern gelten für das Mini-Memory. Wir haben also für den Aufruf des Unterprogramms eine feste Adresse eingegeben und dies ebenfalls beim Wert für den Sprung getan. Hier mußten wir uns während des Programmierens merken, auf welche Adresse wir springen wollten. Weiter haben wir uns die Adresse des Programmbeginns (>2520 bzw. >7D20) und der ersten freien Adresse hinter dem Programm gemerkt (>253A bzw. >7D3A);
Jetzt können wir aber das Programm noch nicht aufrufen und so müssen wir die dafür notwendigen Zeiger noch entsprechend laden.

Zuerst müssen wir das Programm gegen Überschreiben schützen und den Zeiger auf den Beginn des freien Rams laden. Beim Extended Basic ist die Adresse >2002 mit >253A und beim Mini-Mem ist die Speicherzelle >701C mit >7D3A zu laden. Am einfachsten geschieht dies auch mit dem Line-by-Line Assembler mit AORG>2002 (>701C für Mini-Mem) und dann DATA >2536 (>7D36).
Nun müssen wir noch die DEFinitionstabelle und den Zeiger darauf laden. Unter Extended-Basic ist dies die Speicherstelle >2004 (>701E für Mini-Mem). Von dem Wert, der bisher in dieser Speicherstelle steht, ziehen wir 8 ab und geben diesen Wert dann ein, also z.B.> 3FF8 bei Extended Basic oder >7FF8 bei Mini-Mem. Die letzte Stelle dieses Zeigers ist immer eine 8 oder eine 0. Dann gehen wir an die Adresse, deren Wert wir gerade eingegeben haben mit z.B. AORG> 3FF8 (>7FF8) und nun wird es kompliziert. Jetzt müssen wir mittels DATA den Namen in ASCII eingeben. Bei dem Namen Start ergibt sich >5354, >4152, >5420. Bitte beachten, der Name ist immer 6 Stellen lang, notfalls mit >20 (entspricht dem Leerzeichen) füllen. Darauf folgt nun die Startadresse des Programms also >2520 bzw.> 7D20 beim Mini-Mem.
Jetzt können wir das Programm mit dem Namen aufrufen und ausführen.

In Extended Basic bzw. im Basic geschieht dies mittels CALL LINK ("START"). Jetzt funktioniert das Programm über LOAD AND RUN des Mini-Mems und des Assembler-Moduls einwandfrei, wir sehen das A deutlich auf dem Bildschirm, aber aus dem Basic oder Extended Basic aufgerufen, sehen wir gar nichts. Das liegt an einer Eigenheit, wie im Basic der Bildschirm aufgebaut ist: Hier sind alle Zeichen mit einem Offset von> 60 abgebildet. Soll das Programm also unter Basic arbeiten, muß die entsprechende Zeile heißen:

```
LI    R1,>A100
```

Übrigens könnt Ihr mir auch gerne schreiben, wenn Ihr ganz besondere Probleme mit der Assembler-Programmierung habt, aber bitte nicht zur Fehlersuche in 20KByte langen Programmen! Auf die Anwendung des Debuggers werden wir später eingehen.

FÜR EXPERTEN

Bleibt noch unser Tip für die Assembler-Könner unter Euch: Im Betriebssystem der RS232-Karte steckt eine Interrupt-Routine, die ein interruptgesteuertes Empfangen von Zeichen erlaubt. Dazu müssen folgende Pointer gesetzt werden: Auf> 8300 befindet sich der Pointer zum PAB-Buffer im VDP, auf> 8302 (Byte) die maximale Datensatzlänge, auf> 8303 (Byte) die Länge des zu empfangenden Datensatzes und auf>8304 (Byte) die aktuelle Länge des Datensatzes. Im Falle eines Empfangserrors ist das letzte empfangene Zeichen> FF. Hat der Datensatz die auf >8303 angegebene Länge erreicht, wird als empfangenes Zeichen> FE in den PAB Buffer geschrieben. Um das Ganze zu aktivieren, wird ganz normal eine Datei eröffnet, nur mit dem Unterschied, daß in der PAB beim OP-Code das 1. Bit (MSB) gesetzt werden muß, also z.B. 80 für OPEN. Benötigt wird das zum Beispiel für Terminal Emulator Programme.

IM RAM LIEGT MANCHES VERSTECKT

Der TMS 9900 besitzt keine internen Register, wie viele anderen Microprozessoren, sondern verwendet dazu 16 Register a 2 Bytes, die irgendwo im RAM liegen können. Auf den Anfang dieses Registersatzes zeigt der Workspacecounter (Arbeitsbereichszeiger). Dieser Workspacecounter kann mit dem Befehl LWPI geladen werden, wie wir es ja schon beim letzten Mal benutzten. Ein LWPI >2500 belegt also den Speicher wie folgt:

```
> 2500 Register 0
> 2502 Register 1
> 2504 Register 2
> 2506 Register 3
> 2508 Register 4
> 250A Register 5
> 250C Register 6
> 250E Register 7
> 2510 Register 8
> 2512 Register 9
> 2514 Register 10
> 2516 Register 11
> 2518 Register 12
> 251A Register 13
> 251C Register 14
> 251E Register 15
```

Prinzipiell können diese Register im Programm beliebig benutzt werden. Da einige aber auch besondere Bedeutung haben, sollten wir vorher schon einige Vereinbarungen treffen. So wird in R11 bei einem Sprung zu einem Unterprogramm mittels BL (Branch and Link, Sprung mit Sicherung der Rückkehradresse des Programmzählers) genau diese Rückkehradresse abgelegt, d.h. wir sollten R11 nicht anderweitig benutzen. R12 ist nur für CRU-Operationen (dazu kommen wir ein andermal) wichtig. Hier muß dann die CRU-Basisadresse enthalten sein. R13, R14 und R15 werden durch den Aufruf eines Unterprogramms mittels BLWP belegt, sind also in unserem Hauptprogramm verfügbar. Weiter kommt noch R0 besondere Bedeutung bei Schiebeoperationen zu, das soll uns aber noch nicht interessieren.

UNTERPROGRAMME

Auch bzw. gerade in Assembler ist eine „Rattenschwanzprogrammierung" ein sehr schlechter und auch unübersichtlicher Programmierstil. Dennoch ist so etwas leider häufig zu sehen. Dabei erleichtern Unterprogramme, vor allem wenn man viel programmiert, die Arbeit.
Für eine bestimmte Lösung nimmt man nur das entsprechende Unterprogramm aus der Bibliothek. Dazu sollte man sich aber vorher die Benutzung der einzelnen Register genau überlegen.
Üblicherweise benutzen wir R0 bis R2, da diese auch viel von den Hilfsroutinen wie VMBW usw. benutzt werden, für Werte, die sich dauernd ändern. Damit wir für eigene Unterprogramme noch ein bißchen mehr zur Verfügung haben, nehmen wir dafür auch noch R3 und R4. R5 und R6 können für Schleifenzähler reserviert werden, R8 und R9 bleiben dann für Werte, die sehr häufig benötigt werden und R10 verwenden wir als weiteren Speicher für den Rücksprung. Dann können wir in zwei Unterprogrammebenen nach folgendem Muster arbeiten:

```
Hauptprogramm:
     ....     BL @XX
     ....
XX   MOV R11,R10
     .
     BL    @YY
     B     *R10
YY   .
     .
     RT
```

In dem Unterprogramm YY darf dann R10 natürlich nicht verändert werden. Die oben angeführte Aufteilung der Register ist natürlich nur als Vorschlag zu werten. Je nach den Notwendigkeiten des Programms wird sie entsprechend geändert.
Übrigens kann man bei einigen der Assemblerprogrammen bei den Registern das R auch weglassen, MOV R11,R10 kann man dann auch schreiben als MOV 11,10. Schreibfaule Leute machen das immer so. Es spart halt etwas Schreibarbeit.
Bei manchen Assemblerprogrammen können Labels, das sind solche Bezeichnungen wie XX und YY in dem Beispiel, nicht verwendet werden. Dann sind die Unterprogramme dem Hauptprogramm bei der Eingabe voranzustellen, damit wir uns jeweils die Startadresse des Unterprogrammes merken und dann im Hauptprogramm einsetzen können, z.B. als BL @>2500.

TEXTE AUF DEN BILDSCHIRM SCHREIBEN

Bei jedem Programm brauchen wir ja den Dialog mit dem Benutzer. Dazu müssen Informationen auf den Bildschirm geschrieben werden. Hier sind dann in Assembler meist endlose Reihen von LI und BLWP zu sehen. Das Ganze läßt sich aber auch in einem Unterprogramm lösen:

```
TEXT1 DATA >0002
                 (Adresse)
      BYTE 16 (Länge)
      TEXT 'Das ist ein
                  Test'
TEXT  MOV   *R1+,R0
      MOVB  *R1+,2
      SRL   R2,8
      BLWP  @VMBW
      RT
```

Im Hauptprogramm muß dann das Unterprogramm wie folgt aufgerufen werden:

```
      .
      .
      LI    R1,TEXT1
      BL    @TEXT
      .
      .
```

Der Text ist wie folgt aufgebaut. Erst kommen zwei Bytes (ein Wort), welches die Bildschirmadresse enthält, wo der Text hingeschrieben werden soll. Dann kommt ein Byte, das die Länge des Textes enthält und dann der Text.
Achtung: Soll das Programm vom Basic heraus aufgerufen werden. so ist der Screenoffset von >60 zu berücksichtigen, d.h. wir müssen alle Bytes einzeln über DATA oder BYTE eingeben mit addiertem Offset. Ähnliches gilt bei Assemblerprogrammen, die die Anweisung TEXT nicht kennen, hier muß ebenfalls einzeln (mit oder ohne Offset) eingegeben werden.
Das Unterprogramm TEXT bereitet zuerst die Register für die Ausführung von VMBW vor. R1 enthält die Startadresse des Textes im Ram. So wird also zuerst R0 mit der Bildschirmadresse geladen. R1 wird dabei automatisch um 2 erhöht. Dann brauchen wir noch in R2 die Anzahl der zu übertragenden Bytes, also die Länge des Textes. MOVB bewegt ein Byte in das höherwertige Byte von R2, SRL R2,8 ist ein Schiebebefehl und schiebt das Byte in das niederwertige Byte von R2. Mit der Hilfsroutine VMBW wird dann der Text in das VDP-

Ram geschrieben. Mit kleinen Änderungen kann dieses Unterprogramm auch in Schleifen verwendet werden, ohne daß im Hauptprogramm dauernd LI R1 steht.

SCHLEIFEN

Einige Leser fragten auch schon nach der Realisierung von FOR-NEXT-Schleifen in Assembler. Nun, das geht ganz einfach:

```
        LI   R5,200
LOOP    .
        .
        DEC R5
        JNE  LOOP
```

Zuerst wird ein Register, hier R5, mit der Anzahl der Schleifendurchläufe geladen. Dann folgt das, was wir in der Schleife machen wollen. Abschließend wird mit dem Befehl DEC der Inhalt von R5 jeweils um 1 erniedrigt. Solange nicht 0 erreicht wird (beim Befehl DEC wird das Equal-Bit im Statusregister des Prozessor gesetzt, wenn das Ergebnis 0 ist), führt der Befehl JNE (Springe wenn nicht gleich) in die Schleife.

DER RÜCKSPRUNG ZU BASIC

Wollen wir am Ende eines Maschinenprogramms wieder zu dem Programm zurückkehren, aus dem das Maschinenprogramm aufgerufen wurde, so ist das ganz einfach zu realisieren:

```
CLR    @ >837C
LWPI     >83E0
       @ >0070
```

Dieser Rücksprung führt in den GPL-Interpreter zurück, wobei zuerst das GPL-Statusbyte gelöscht wurde, um anzuzeigen, daß kein Fehler aufgetreten ist. Dann wird wieder der GPL-Workspace geladen und das Programm, welches die Maschinenroutine aufgerufen hat, weiter ausgeführt.

FÜR EXPERTEN: AUTOSTART IN X-BASIC

Der Editor/Assembler und das Mini-Memory kennen einen Autostart des Maschinenprogramms direkt danach, wenn es von der Diskette geladen worden ist. Der Lader des Extended Basic kennt diesen Befehl leider nicht. Er läßt sich aber einfach nachbilden: Der Trick funktioniert über den User defined Interrupt. Die letzten Zeilen unseres Assembler-Programms (nicht mit einem Line-by-Line-Assembler erstellt) heißen dann:

```
AORG >83C4
DATA  START
```

Damit wird als letztes im Ladevorgang die anwenderdefinierte Interrupt-Routine mit der Startadresse unseres Maschinenprogramms geladen. Wenn jetzt ein Interrupt ausgeführt wird, was ja alle 50stel Sekunde erfolgt, wird das Programm ausgeführt. In unserem Programm müssen wir dann nur ganz am Anfang noch einfügen:

```
CLR @>83C4
```

Damit nicht aus Versehen, wenn wir Interrupts zulassen, wieder von vorne begonnen wird.



EIN BISSCHEN BIT BITTE

Eine Anregung, etwas näher auf das Statusregister des TMS 9900 einzugehen, möchte ich aber gerne aufgreifen. Zuvor aber noch etwas anderes: Die Hilfsroutinen. Prinzipiell lassen sich beim TI 99/4A alle Maschinenprogramme auch unter allen Modulen, mit denen dies möglich ist, einsetzen. Eventuell müssen sie in einen anderen Speicherbereich gelegt werden, aber das ist nur bei einem Assembler, der mit absoluter Adresse arbeitet, wichtig. Ein Hindernis sind aber die von den einzelnen Modulen zur Verfügung gestellten Hilfsroutinen, die mittels des Befehls BLWP ausgeführt werden. Sie,bzw. die Vektoren darauf, stehen bei jedem Modul woanders. In der Tabelle 1 sind die Adressen zum Vergleich. Dabei ist nur zu berücksichtigen, daß das XMLLNK des Extended-Basic anders funktioniert als bei den beiden anderen Modulen.

Wollen wir also bei einem Programm ein Byte in das VDP-Ram schreiben, so ist bei einem Programm für das Mini-Memory zu schreiben:

```
    LI      R0,>0100(VDP–Adresse)
    LI      R1,>3100(BYTE)
    BLWP@>6024
```

Für das Extended Basic müßte dies in einer Zeile geändert werden:

```
    BLWP@>2020
```

Für das Editor/Assembler-Modul ist dann noch eine andere Adresse notwendig. Hier arbeiten wir aber besser mit der REFerence. Dieses REF bedeutet, daß beim Laden des Maschinenprogramms von Diskette eine REF-Tabelle nach dem Begriff, der in der Definition hinter REF genannt wird, abgesucht und der in dieser Tabelle enthaltene Wert dafür im Maschinenprogramm eingesetzt wird. Für das E/A-Modul schreiben wir also an den Programmanfang in den Quellcode:

```
    REF    VSBW
```

und dann

```
    BLWP@VSBW
```

Dies hat den Vorteil, daß dieses Programm dann auch ohne Änderungen für das Mini-Memory verwendet werden kann, da auch das Mini-Memory über die REFerence-Tabelle verfügt und beim Laden des Maschinenprogramms von Diskette dann eben die fürs Mini-Mem richtigen Werte eingesetzt werden. Leider geht das nicht fürs Extended Basic.

DAS EQUAL-BIT IM STATUSREGISTER

Kommen wir nun zum Statusregister des TMS 9900, wobei gleich vorab gesagt sei, daß sich dieses Thema nicht in einer Folge erläutern läßt, wir also immer wieder darauf zurückkommen werden. Der TMS 9900 besitzt neben dem Workspacecounter, den wir das letzte Mal besprochen haben, einen Programmcounter, das ist ein Register, welches die Adresse des nächsten abzuarbeitenden Befehls enthält, und ein Statusregister. Dieses wird durch das Ergebnis von vielen der Befehle beeinflußt, es werden je nach Ergebnis einzelne Bits gesetzt. Heute soll uns nur einmal das Bit 2 interessieren, auch als Equal-Bit bezeichnet. Der Name sagt es schon, dieses Bit wird gesetzt, wenn irgend etwas „Gleich" ist. Dies gilt für alle Vergleiche, also C (Compare = Vergleiche), CB (Vergleiche Byte) und COC, CZC und TB (diese sind etwas komplizierter, zu diesen kommen wir aber später noch). Bei allen anderen Operationen wird das Ergebnis 0 verglichen und wenn es 0 ist, dieses Bit ebenfalls gesetzt.

Betrachten wir also nochmal die Schleife:

```
    LI    R5,200
LOOP .
     .       DEC R5
    JNE LOOP
```

Der Befehl DECrement erniedrigt jeweils den Inhalt von R5 um 1. Erreicht dieser 0, wird das Statusbit für Equal gesetzt und der nachfolgende Sprung-Befehl JNE (Jump not equal) unbeachtet gelassen. Daraus sehen wir deutlich, daß die folgendes

```
    DEC     R5
    CI      R5,>0000
    JNE     LOOP
```

zwar das gleiche Ergebnis bringt, aber eben einen überflüssigen Befehl enthält. Hier wird durch den (eben überflüssigen) Befehl CI (Vergleiche unmittelbar) der Inhalt von R5 mit 0 verglichen.

AUSGABE EINER HEX-ZAHL AUF DEN BILDSCHIRM

Ein weiteres Beispiel soll die Darstellung einer hexadezimalen Zahl in Dezimal auf dem Bildschirm sein. In Maschinenprogrammen wird ja häufig nur mit Ganzzahlen gerechnet, meist mit einem Registerinhalt. Mittels folgendem Unterprogramm kann dies realisiert werden:

```
     LI    R0,>0100 (Adresse)
     LI    R3,>0300 (Zahl)
     BL @HEXD
D10 DATA >000A
HEXD CLR    R2
     DIV   @D10,R2
```

```
MOV    R3,R1
SWPB   R1
AI     R1,>3000
BLWP   @ VSBW
DEC    R0
MOV    R2,R3
JNE    HEXD
B
```

Wir finden darin einen neuen Befehl, DIV (divide = teilen). Bei diesem ist zu beachten, daß für den Senkenoperanden 2 Register zur Verfügung stehen müssen, deshalb wird zuerst R2 auf 0 gesetzt mit CLR (Clear = Löschen). Dann erfolgt die Division durch 10. Als Ergebnis steht nun in R2 der Quotient und in R3 der Rest. Diesen Rest müssen wir nun als Zahl auf den Bildschirm darstellen. Um die Routine VSBW benutzen zu können, muß in R0 die Bildschirmadresse, das wurde vom Hauptprogramm erledigt, und im höherwertigen Byte von R1 das zu schreibende Byte stehen. Dies wird durch MOV und DWPB (Swap Byte = tausche Byte) erledigt. Nun müssen wir mit AI (Unmittelbare Addition) noch einen lesbaren ASCII-Wert daraus machen. Für Programme, die aus dem Basic gestartet werden, muß hier >9000 stehen. Danach folgt die neue Bildschirmadresse (nächste Stelle der Zahl), und dann werden mit MOV wieder die Register für die Division vorbereitet. Der Trick dabei ist, daß auch bei MOV das Equal-Bit des Statusregisters gesetzt wird, wenn das Ergebnis, d.h. hier der Inhalt von R3, Null ist. Gibt es also nichts mehr, durch das geteilt werden könnte, dann wird der nachfolgende Sprung nicht mehr ausgeführt und es erfolgt der Rücksprung mit B.

Für Experten:
Kassettenzugriff aus Assembler

In einer TI-Ausgabe hatten wir bei der Beantwortung von Leserfragen angegeben, daß es ein Leichtes sei, aus Assembler heraus die Adventure-Spiele auf Diskette zu überspielen. Wie mehrere Anrufe und Briefe deuten, gibt es da aber auch bei erfahrenen Programmierern Probleme. Es scheint auch so, daß hier einmal in einem Buch eine falsche Routine veröffentlicht wurde. Also, die Kassettenroutine ist eigentlich im Assembler-Handbuch und im Minimem-Handbuch ausreichend beschrieben. Wichtig ist nur noch eines: Auf FAC (>834A) muß der Name in ASCII stehen, also z.B. CS1. Das kann wie folgt realisiert werden:

```
NAME   TEXT'CS1'
       MOV   @NAME,@FAC
       MOVB  @NAME+2@FAC
```

Ansonsten braucht man sich nur an die Anweisungen in den genannten Handbüchern zu halten.

Tabelle 1:
Equates für die Hilfsroutinen:

Bez.	E/A	Mini.mem	Ex-Basic	Beschr.
VSBW	>210C	>6024	>2020	Schreibt ein Byte in VDP-Ram
VMBW	>2110	>6028	>2024	Schreibt mehrere Bytes in VDP-Ram
VSBR	>2114	>602C	>2028	Liest ein Byte aus VDP-Ram
VMBR	>2118	>6030	>202C	Liest mehrere Bytes aus VDP-Ram
VWTR	>211C	>6034	>2030	Schreibt in VDP-Register
KSCAN	>2108	>6020	>201C	Tastaturabfrage
XMLLNK	>2104	>601C	>2018	Ausführen einer Betriebssystem-Routine
DSRLNK	>2120	>6038	n.a.	Aufrufen einer DSR-Routine
GPLLNK	>2100	>6018	n.a.	Ausführen einer GPL-Routine
LOADER	>2124	>603C	n.a.	Laden von Assembler-Programmen
NUMASG	n.a.	>6040	>2008	Wert an num. Basicvariable
NUMREF	n.a.	>6044	>200C	Wert von num. Basicvariable holen
STRASG	n.a.	>6048	>2010	Wert an String-Basicvariable
STRREF	n.a.	>604C	>2014	Wert von String-Basicvariable
ERR	n.a.	>6050	>2034	Rueckkehr Basic mit Error

Assembler leicht gemacht

Zur Erinnerung wollen wir hier das Beispielprogramm nochmals wiederholen. Es diente zur Ausgabe einer Hex-Zahl bzw. Registerinhaltes als Dezimalzahl auf dem Bildschirm.
Wie schon erklärt, finden wir darin einen neuen Befehl:

```
LISTING 1:

        LI    RØ,>Ø1ØØ
        LI    R3,>Ø318
        BL    @HEXD


D1Ø     DATA  >ØØØA

HEXD    CLR   R2
        DIV   @D1Ø,R2
        MOV   R3,R1
        SWPB  R1
        AI    R1,>3ØØØ
        BLWP  @VSBW
        DEC   RØ
        MOV   R2,R3
        JNE   HEXD
        B     *11
```

DIV (divide=teilen). Bei diesem ist zu beachten, daß für den Senkenoperanden 2 Register zur Verfügung stehen müssen, deshalb wird zuerst R2 auf 0 gesetzt, mit CLR (Clear=Löschen) ; in R3 ist ja die Hexzahl, die ausgegeben werden soll, enthalten. Dann erfolgt die Division durch 10. Diesen Wert stellen wir als getrenntes DATA zur Verfügung. Als Ergebnis steht nun in R2 der ganzzahlige Quotient und in R3 der Rest. Diesen Rest müssen wir nun als Zahl auf den Bildschirm darstellen.
Um die Routine VSBW benutzen zu können, muß in R0 die Bildschirmadresse stehen, das wurde vom Hauptprogramm erledigt, und im höherwertigen Byte von R1 das zu schreibende Byte. Dies wird durch MOV und SWPB (Swap Byte=tausche Byte) erledigt. Nun müssen wir mit AI (Unmittelbare Addition) noch einen lesbaren ASCII-Wert daraus machen. Für Programme, die aus dem Basic gestartet werden, muß hier >9000 stehen, um den Screenoffset auszugleichen. Danach folgt die neue Bildschirmadresse (nächste Stelle der Zahl), und dann werden mit MOV wieder die Register für die Division vorbereitet. Der Trick dabei ist, daß auch bei MOV das Equal-Bit des Statusregisters gesetzt wird, wenn das Ergebnis, d.h. hier der Inhalt von R3 Null ist. Gibt es also nichts mehr, durch das geteilt werden könnte, dann wird der nachfolgende Sprung nicht mehr ausgeführt und es erfolgt der Rücksprung mit B.
Ein weiterer Punkt, der häufig zu Unsicherheiten führt, ist die Darstellung negativer Zahlen, wie sie der TMS 9900 verwendet. Manch einer wird sich auch schon gefragt haben, warum bei CALL PEEK und CALL LOAD manchmal eine negative Zahl als Adressenangabe zu verwenden ist. Dies hat seine Begründung in der Darstellung von negativen Zahlen für die CPU.
Hier müssen wir eine kleine Ausführung über das Zahlensystem unseres Computers einfügen. Bekanntlich kann der Rechner nur Einsen und Nullen (binäres System) unterscheiden, während wir im sogenannten Dezimalsystem rechnen. Nun lassen sich diese Zahlen einfach umrechnen. Darüber gibt die Tabelle 1 Aufschluß. Darin sind ebenfalls die Hexadezimalen Zahlen aufgeführt. Diese sind quasi aus der Notwendigkeit entstanden, nicht mit ewig langen Kolonnen von binären Zahlen arbeiten zu müssen. Ein Byte enthält bekanntlich 8 Bit und statt 00011010 schreibt sich wohl besser >1A, dabei dient das Größer-Zeichen zur Erkennung als Hexadezimalzahl. Als 16 Bit-Prozessor arbeitet der TMS 9900 sogar mit 16 Bit (2 Byte) langen Worten, das ergibt dann eine vierstellige Hexadezimale Zahl.

Tabelle 1:

Binär	Hex	Dezimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Kommen wir aber zu dem Vorzeichen der Zahl zurück. In nahezu allen Befehlen arbeitet die CPU mit folgender Zahlendarstellung: Das 1. Bit (MSB =höchstwertigstes Bit) der insgesamt 16 Bit langen Zahl bestimmt das Vorzeichen. Ist es 0, so handelt es sich um eine positive Zahl. Ist es 1, dann ist sie negativ. Dabei ist die negative Zahl das Zweier-Komplement der positiven Zahl. Einige Beispiele:

>0000 = 0
>0001 = 1
>7FFF= 31767
>8000 = 32768 (erstes Bit gesetzt!)
>FFFE= −2
>FFFF= −1

Wozu dient nun das Alles? Nun, das Statusregister des TMS 9900 hatten wir schon erwähnt und uns bisher ausschließlich mit dem Equal-Bit in diesem Register beschäftigt. Dieses Equal-Bit ist immer dann auf 1 gesetzt, wenn das Ergebnis einer Operation 0 war oder wenn bei einem Vergleich die Operanden gleich waren. In dem Statusregister beinhalten aber nach den meisten Operationen – (es gibt auch wenige, die das Statusregister nicht beeinflussen, darüber gibt aber das Handbuch zum Editor/Assembler Modul Auskunft) – noch andere Bits wichtige Informationen. Zwei davon wollen wir uns heute noch etwas näher ansehen: Das Logical Greater Bit (Logisch größer) und das Arithmetic Greater Bit (arithmetisch größer). Bleiben wir zuerst beim Letzteren, dem Arithmetisch Größer Bit. Wie der Name schon sagt, wird hier ein Vergleich nach den arithmetischen Regeln ausgeführt, d.h. das Vorzeichen der Zahl wird entsprechend berücksichtigt. Mit der Vergleichsoperation CI (Vergleiche unmittelbar) läßt sich die Auswirkung am einfachsten zeigen:

Der Befehl LI beeinflußt übrigens das Statusregister (im Gegensatz zu CLR), sodaß bei den letzten

ASSEMBLER-KURS

```
LISTING 2:
        LI    R1,>1ØØØ   *DEZ 4Ø96
        CI    R1,>Ø5ØØ   *DEZ 128Ø

ARITHMETRIC GREATER BIT gesetzt

        LI    R1,>Ø48Ø   *DEZ 1152
        CI    R1,>Ø5ØØ   *DEZ 128Ø

ARITHMETRIC GREATER BIT nicht gesetzt

        LI    R1,>FFFE   *DEZ -1
        CI    R1,>FØØØ   *DEZ -4Ø96

ARITHMETRIC GREATER BIT gesetzt

        LI    R1,>EØØØ   *DEZ -8192
        CI    R1,>FØØØ   *DEZ -4Ø96

ARTHMETRIC GREATER BIT nicht gesetzt

        LI    R1,>FFFE   *DEZ -1
        CI    R1,>Ø5ØØ   *DEZ 128Ø

ARITHMETRIC GREATER BIT nicht gesetzt

        LI    R1,>1ØØØ   *DEZ 4Ø96
        CI    R1,>FØØØ   *DEZ -4Ø96

ARTHMETRIC GREATER BIT gesetzt

        LI    R1,>EØØØ   *DEZ -8192
        MOV   R1,R1

ARTHMETRIC GREATER BIT nicht gesetzt

        CLR   R1         *DEZ Ø
        MOV   R1,R1

ARITHMETRIC GREATER BIT nicht gesetzt

        LI    R1,>1ØØØ   *DEZ 4Ø96
        MOV   R1,R1

ARTHMETRIC GREATER BIT gesetzt
```

Beispielen die MOV-Operation entfallen kann. Bei dieser wird ja der Registerinhalt des Senkenoperanten mit 0 verglichen und so soll hier nur gezeigt werden, wie auf einfache Art immer die entsprechenden Statusbits gesetzt werden können.
Für dieses Arithmetric Greater Bit stehen uns nun zwei bedingte Sprünge zur Verfügung. Bei JGT (Jump greater than) wird der Sprung ausgeführt, wenn das Bit gesetzt ist und bei JLT (Jump less than) wird der Sprung ausgeführt, wenn das Bit nicht gesetzt ist und das Equal-Bit ebenfalls 0 ist.
Kommen wir also zu dem Logical Greater Bit. Dieses Bit wird im Statusregister immer dann gesetzt, wenn etwas logisch größer ist, d.h. ohne Beachtung eines Vorzeichens, das 16. Bit gehört dabei zur Zahl.

Selbstverständlich gibt es auch einige Sprungbefehle, die vom Logical greater Bit beeinflußt werden.
Bei JH (Jump high) wird der Sprung ausgeführt, wenn dieses Bit gesetzt ist, bei JL (Jump Low) wenn es nicht gesetzt wird. JHE (Jump high or equal) und JLE (Jump low or equal) berücksichtigt dann jeweils auch das Equal-Bit.

Auch hier wieder einige Beispiele zum besseren Verständnis:

```
LISTING 3:
        LI    R1,>1ØØØ   *DEZ 4Ø96
        CI    R1,>Ø5ØØ   *DEZ 128Ø

LOGICAL GREATER BIT gesetzt

        LI    R1,>Ø48Ø   *DEZ 1152
        CI    R1,>Ø5ØØ   *DEZ 128Ø

LOGICAL GREATER BIT nicht gesetzt

        LI    R1,>FFFE   *DEZ -1
        CI    R1,>FØØØ   *DEZ -4Ø96

LOGICAL GREATER BIT gesetzt

        LI    R1,>EØØØ   *DEZ -8192
        CI    R1,>FØØØ   *DEZ -4Ø96

LOGICAL GREATER BIT nicht. gesetzt

        LI    R1,>FFFE   *DEZ -1
        CI    R1,>Ø5ØØ   *DEZ 128Ø

LOGICAL GREATER BIT gesetzt

        LI    R1,>1ØØØ   *DEZ 4Ø96
        CI    R1,>FØØØ   *DEZ -4Ø96

LOGICAL GREATER BIT nicht gesetzt

        LI    R1,>EØØØ   *DEZ -8192
        MOV   R1,R1

LOGICAL GREATER BIT gesetzt

        CLR   R1         *DEZ Ø
        MOV   R1,R1

LOGICAL GREATER BIT nicht gesetzt

        LI    R1,>1ØØØ   *DEZ 4Ø96
        MOV   R1,R1

LOGICAL GREATER BIT gesetzt

     -      ) UUUUUUUUUU
```

ASSEMBLER-KURS

Auf der vorigen Seite mußte der kleine Tip für die Experten entfallen. Das waren dann die berühmten 10 Zeilen zuviel beim Umbruch, d.h. 10 Zeilen wären über die Seite hinausgegangen und „Gummi-Seiten" gibt es leider noch nicht.
Deshalb diesen Tip nun hier am Anfang: Bei der Dateibehandlung unterscheidet der TI 99/4A ja 5 verschiedene Formate: Program, DIS/VAR, DIS/FIX, INT/VAR und INT/FIX. Gemäß dem Handbuch zum Editor-Assembler kann über die Abfrage des Status einer Datei auch das Format in Erfahrung gebracht werden. Dazu wird einfach ganz normal die Datei über DSRLNK angesprochen. Der OpCode im PAB ist dafür dann >09. Hier ist in den meisten Disk-Controlern ein Softwarefehler enthalten. DIS/FIX-Dateien werden bei der Statusabfrage als DIS/VAR zurückgegeben.
Nun wollen wir aber weiter in unserem Assemblerkurs fortfahren. Viele Anfragen erhielt ich nach der Belegung des VDP-Ram's, d.h. was steht dort wo. Dies soll dann auch heute unser Thema sein. Vorab muß aber dazu gesagt werden, daß sich die Belegung je nach Modul unterscheidet. Wir wollen uns heute auf TI-Basic, Extended Basic und das Editor/Assembler-Modul unter der Funktion LOAD AND RUN beschränken, dabei ist das Mini-Memory unter LOAD AND RUN gleich wie das Assembler-Modul.
Rein für die Bildschirmdarstellung braucht der Video-Display Prozessor einige Tafeln. Im einzelnen sind dies:
1. Screen Image Table, auch als Bildschirmtabelle zu bezeichnen. In diesem Bereich stehen die abzubildenden Zeichen in ASCII. Dabei ist die linke obere Ecke des Bildschirms der Anfang der Tafel, der Buchstabe rechts daneben das nächste Byte usw. Das 33. Byte ist also das erste Zeichen in der zweiten Zeile. Insgesamt benötigt diese Pattern Descriptor Table 768 Bytes (24 Zeilen a 32 Spalten).
2. Die Color Table, also die Farbtabelle. Diese bestimmt die Farben der Zeichen jedes Charaktersatzes (jeweils 8 Zeichen, wie in CALL

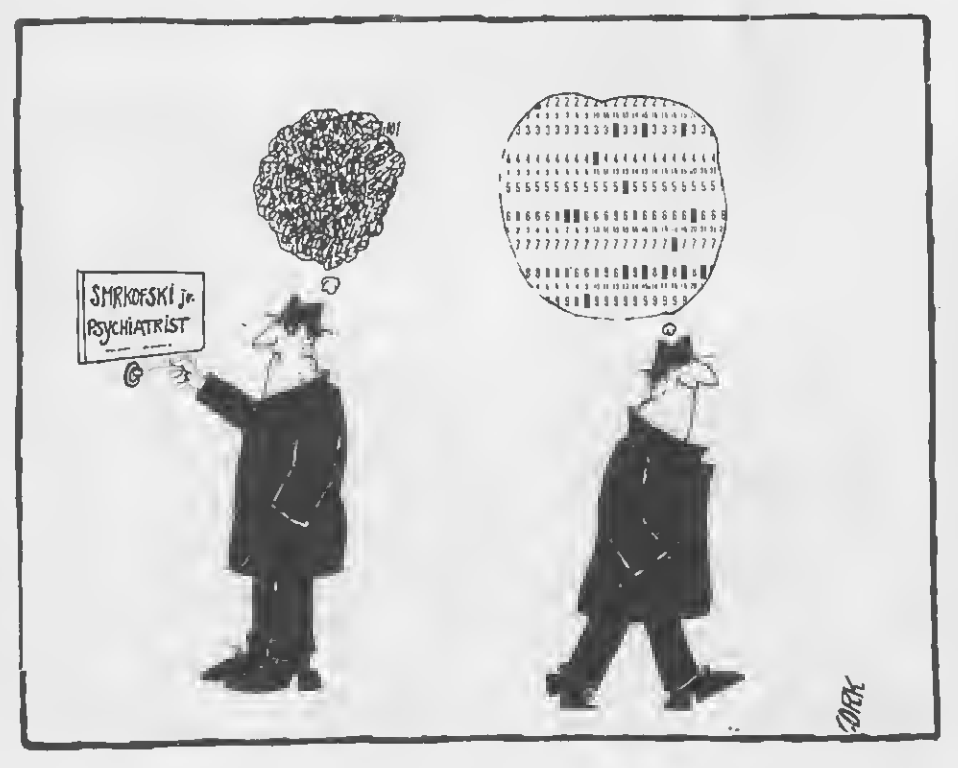

```
****************************
* VDP-REGISTER UND TABELLEN LADEN
*

SETREG LI   1,REG          * REGISTER WERTE
REG1   MOV  *1+,Ø          * WERT AUS DER TABELLE
       JEQ  REGEND
       BLWP §VWTR          * REGISTER LADEN
       JMP  REG1
REGEND MOVB §REG+1,§>83D4 * KOPIE VDP-REG 1 FUER INTERRUPT
       LI   Ø,>Ø3ØØ        * >DØ AUF BEGINN DER SPRITE
       LI   1,>DØØØ        * ATTRIBUTE TABLE, KEINE SPRITES
       BLWP §VSBW          * WERDEN ABGEBILDET
       LI   Ø,>Ø38Ø        * COLOR TABLE
       LI   1,>FØØØ        * VORDERGRUND WEIS AUF TRANSPARENT
REG2   BLWP §VSBW
       INC  Ø
       CI   1,>Ø3AØ
       JNE  REG2
       LI   1,>Ø9ØØ        * BEGINN AB CHARAKTER >2Ø = SPACE
       MOV  1,§>834A
       BLWP §GPLLNK        * GR. BUCHSTABEN LADEN
       DATA >ØØ18
       BLWP §GPLLNK        * KL. BUCHSTABEN LADEN
       DATA >ØØ4A
       LI   Ø,>Ø8FØ        * PATTERN DESCRIPTOR TABLE
       LI   1,CURSOR       * CURSOR CHAR
       LI   2,>ØØ1Ø        * 16 BYTES
       BLWP §VMBW
       RT

CURSOR DATA >3C42,>99A1,>A199,>423C * COPYRIGHT ZEICHEN
       DATA >3C3C,>2424,>2424,>3C3C * CURSOR

* DATEN FUER DIE VDP-REGISTER


REG    DATA >Ø1EØ  * REG. 1: 16KBYTE, GRAPHICS MODE
       DATA >Ø2ØØ  * REG. 2: BASIS SCREEN IMAGE TABLE
*                  * WERT MAL >4ØØ (HEX!)
       DATA >Ø3ØE  * REG. 3: BASIS COLOR TABLE
*                  * WERT MAL >4Ø (HEX!) >4Øx>ØE=>38Ø
       DATA >Ø4Ø1  * REG. 4: BASIS PATTERN DESCRIPTOR TABLE
*                  * WERT MAL >8ØØ (HEX) >8ØØx>Ø1=>8ØØ
       DATA >Ø5Ø6  * REG. 5: BASIS SPRITE ATTRIBUTE LIST
*                  * WERT MAL >8Ø (>8Øx>Ø6=>3ØØ)
       DATA >Ø6ØØ  * REG. 6: SPRITE DESCRIPTOR TABLE
*                  * WERT MAL >8ØØ
       DATA >Ø7F4  * REG. 7: FARBE FUER TEXTMODE GRAU
*                  * UND HINTERGRUND DUNKELBLAU
       DATA >ØØØØ  * KENNWERT ENDE DER TABELLE
```



COLOR im Basic, nur stimmen die Nummern der Zeichensätze nicht überein, da der Video-Display-Prozessor von 0 für die ASCII-Character 0 – 7 rechnet). Für jeden Charaktersatz brauchen wir hier ein Byte, insgesamt also 32 Bytes (256 geteilt durch 8). In jedem Byte bestimmen die ersten vier Bits (jeweils vier Bits werden auch als Nybble bezeichnet, hier ist also das höchstwertige Nybble gemeint), die Vordergrundfarbe und das niedrigwertige Nybble die Hintergrundfarbe. Auch hier gelten wieder die Werte aus dem Basic für die Farben, nur müssen wir 1 davon abziehen, also ergibt sich 0 für Transparent und >F bzw. 15 für Weiß.

3. Die Pattern Descriptor Table, die Tafel zur Bestimmung der Zeichen. Diese wird aus dem Basic mit CALL CHAR geladen und ist eigentlich für jedes Zeichen genauso aufgebaut wie der 16stellige String, den wir in Basic eingeben. Dieser String kann ja auch als 8Bytes Hexadezimal gesehen werden. Diese Tabelle umfaßt also 256 mal 8 Bytes, macht 2048 Bytes. Um Platz zu sparen, werden aber im Basic nicht alle 256 Zeichen die der Video-Prozessor darstellen kann ausgenutzt, so daß die Tabelle dann kleiner gehalten ist.

4. Die Sprite Attribute Tabelle, also die Sprite Werte Tabelle. Hier befinden sich jeweils 4 Bytes für jeden Sprite. Das erste bezeichnet die Lage in vertikaler Richtung (>FF ist ganz oben, dann folgt >00 bis >BE nach unten) und das zweite Byte bestimmt die Lage in horizontaler Richtung (>00 ist ganz links, >FF rechts). Das nächste Byte legt den Charakter-Code fest und das letzte Byte bestimmt die Farbe des Sprite im niedrigwertigen Nybble. Vom höherwertigen Nybble wird nur das letzte Bit verwendet. Ist es nicht gesetzt, ist die obere linke Ecke des Sprites mit den Angaben für die Position bestimmt und der Sprite wandert auf der rechten Seite des Bildschirmes sauber heraus. Ist dieses Bit gesetzt, dann ist die rechte obere Ecke bestimmt, d.h. der Sprite wird um 32 Pixel nach links verschoben und wandert auf der linken Seite richtig heraus. Der Video-Prozessor kann 32 Sprites verwalten, also ist diese Tabelle insgesamt 128 Bytes lang, aber auch hier wird im Extended Basic etwas Platz gespart und nur 28 Sprites werden verwendet.

5. Die Sprite Motion Table, Sprite Bewegungs-Tabelle. Diese Tabelle wird eigentlich nicht vom Video-Prozessor benötigt, sondern wird ausschließlich von der Interrupt-Routine verwaltet. Je Sprite sind auch hier wieder 4 Bytes vorhanden. Die ersten beiden geben den Geschwindigkeitswert in vertikaler und horizontaler Richtung an. Negative Zahlen geben dabei eine umgekehrte Bewegung. Die beiden weiteren Bytes werden nur von der Interrupt-Routine zur zwischenzeitlichen Speicherung von Werten benutzt.

6. Die Sprite Descriptor Table. Diese Tabelle ist genauso wie die Pattern Descriptor Table aufgebaut, nur gilt sie eben für die Sprites. Häufig wird sie auch mit der Pattern Descriptor Table zusammengelegt, so auch in Extended Basic. So brauchen wir darauf hier nicht noch weiter einzugehen.

Was liegt nun von diesen Tabellen wo im VDP-RAM. Zuerst unter TI-Basic:

VDP-Adresse	Tabellenbezeichnung
HEX DEZ	
>0000 0	
	Screen Image Table
>02FF 767	
>0300 768	
	Color Table
>031F 799	
>0320 800	
>03EF 1007	
	Pattern Descriptor Table mit Offset 96 (HEX> 60)Start daher eigentlich ab> 0000
>03F0 1008	Start mit Charakter für ASCII 30
>07FF 2047	Ende variable je nach definierten Zeichen

Der Bereich von> 0320 bis> 03BC wird vom TI-Basic zum Umwandeln von Eingabezeilen in das Basic-Format (Crunchen) verwendet.

Nun zum Extended Basic:

VDP-Adresse	Tabellen-Bezeichnung
HEX DEZ	
>0000 0	
	Screen Image Table
>02FF 767	
>0300 768	
	Sprite Attribut Table
>036F 879	
>0370 880	
>03EF 1007	
	Pattern Descriptor Table mit Offset 96 (HEX> 60)Start daher eigentlich ab> 0000
>03F0 1008	Start mit Charakter für ASCII 30
>077F 1919	
>0780 1920	
	Sprite Motion Table
>07FF 2047	
>0800 2048	
	Color Table
>081F 2079	

Der Bereich von >0370 bis >03EF und> 08C0 und >0967 wird vom Extended Basic zur Zwischenspeicherung von Werten und zum Umwandeln von Eingabezeilen benutzt. Das Editor-Assembler-Modul nimmt weniger Rücksicht, braucht es ja auch gemäß seiner Bestimmung nicht. Es ergibt sich folgende Aufteilung:

VDP-Adresse	Tabellen-Bezeichnung
HEX DEZ	
>0000 0	
	Screen Image Table
>02FF 767	
>0300 768	
	Sprite Attribut Table
>037F 895	
>0380 896	
	Color Table >039F 927
>03A0 928	
>03FF 1023	
>0400 1024	
	Sprite Descriptor Table nur Zeichen ab> 80 (128 dez.) bis> EF (dez 239) benutzbar.
>077F 1919	
>0780 1920	
	Sprite Motion Table
>07FF 2047	
>0800 2048	
	Pattern Descriptor Table ohne Offset für 256 Charakter
>0FFF 4095	
>1000 4096	

Im Basic befinden sich nun oberhalb dieser Tabellen im VDP-RAM das Programm, die Werte aller Variablen und noch einige „Kleinigkeiten". Beim Editor-Assembler ist der Platz frei. Bei einem angeschlossenen Disk-Laufwerk ist ganz am oberen Ende des VDP-RAM's noch ein Speicher reserviert, in dem nichts verändert werden sollte. Zu diesem Speicherbereich werden wir, wie auch zum Aufbau eines Basicprogrammes, in einem anderen Beitrag kommen. Heute soll am Schluß ein kurzes Beispielprogramm stehen, wie alle notwendigen Tafeln des VDP-RAM's und die Register des Video-Prozessors geladen werden können. Die Kommentare geben auch Aufschluß über die Bedeutung der VDP-Register.

GRÖSSER UND KLEINER JETZT WIRD ES TRICKREICH

Jedes Modul, welches beim TI 99/4A Maschinensprache ermöglicht, stellt uns ja einige Hilfsroutinen für die Assemblerprogramme zur Verfügung, auf die wir hier zum Teil näher eingehen wollen.

Als erstes sollen die Routinen zum Beschreiben und Lesen des VDP-RAM's erwähnt werden, also VSBW, VSBR, VMBW, VMBR und VTWR. Nehmen wir die letzte Routine zuerst. Diese dient dazu, in ein bestimmtes Register des Video-Prozessors einen Wert zu schreiben. In R0 unseres Workspaces (Arbeitsbereiches) muß dabei im höherwertigen Byte die Nummer des Registers, also z.B. >01 für Register 1 und im niederwertigen Byte der Wert, der geschrieben werden soll, also z.B. >E0 stehen. Zu beachten ist hier, daß auf die Speicherstelle >83D4 immer eine Kopie des Wertes von VDP-Register 1 stehen muß. Das folgende Listing 1 zeigt, wie das gemacht werden kann:

```
* Listing 2:

       REF   VSBW,VSBR *Fuer Ass. u. MM

VSBW   EQU   >2020  * Fuer X-BAsic
VSBR   EQU   >2028  * Fuer X-BAsic

* Byte schreiben:
       LI    R0,>0190
       LI    R1,>4100   * >A100 Fuer X-Basic
       BLWP  @VSBW

* Byte lesen:
       LI    R0,>0190
       BLWP  @VSBR
* In R1 jetzt das gelesene Byte
```

```
* Listing 1:

       REF   VWTR  * Fuer Assembler u. MM

VWTR   EQU   >2030 * Fuer X-Basic

       LI    R0,>01E0
       BLWP  @VWTR
       LI    R0,>E000
       MOVB  R0,@>83D4

* Oder "getrickst":

       LI    R0,>01E0
WERT   EQU   $-1
       BLWP  @VWTR
       MOVB  @WERT,@>83D4
```

Der Trick im zweiten Teil des Listings bezieht sich auf eine besondere Möglichkeit mancher TMS 9900 Assembler. Das Dollarzeichen wird als Wert des augenblicklichen Standes des Programmzählers genommen. Hier wird dem Label XYZ also die Adresse des Bytes, in dem >E0 steht, zugeordnet. Ein Beispiel, wie VWTR benutzt wird, hatten wir ja schon in der letzten Ausgabe besprochen.

Die anderen Hilfsroutinen haben wir in den letzten Folgen auch schon verwendet. Dennoch wollen wir erst einmal die Abkürzungen erklären. Das V steht dabei für das VDP-RAM, S steht für Single, also Einzeln, M für Multiple, also mehrfach, B für Byte, W für Write (Schreiben) und R für Read, also lesen. VMBW steht also für mehrere Bytes in das VDP-RAM schreiben.
Die Routinen VSBW und VSBR beziehen sich also auf einzelne Bytes. Beiden gemeinsam ist, daß vor dem Aufruf R0 unser Workspace die VDP-Adresse enthalten muß, auf die wir schreiben bzw. von der wir ein Byte lesen wollen. Im höherwertigen Byte von R1 muß der Wert enthalten sein, den wir schreiben wollen, bzw. bei VSBR enthält das höherwertige Byte von R1 den Wert, den wir gelesen haben. Das Listing 2 zeigt nochmal die Routine aus der ersten Folge zum Darstellen eines A auf dem Bildschirm und danach lesen wir wieder den Inhalt der gleichen Speicherstelle.

```
* Listing 3:

* Fuer Assembler und Mini-Mem

        REF   VMBW,VMBR

TEXT1   TEXT 'DAS IST EIN TEST'

        LI    R0,>00A1
        LI    R1,TEXT1
        LI    R2,>0010
        BLWP  @VMBW

* Fuer X-Basic:

VMBW    EQU   >2024
VMBR    EQU   >202C

TEXT1   BYTE >B4,>A5,>B3,>B4

        LI    R0,>00A1
        LI    R1,TEXT1
        LI    R2,>0004
        BLWP  @VMBW

* Bildschirm abspeichern:

        DEF   VSAVE,VLOAD

        REF   VMBW,VMBR * Fuer Ass. u. MM.

VMBW    EQU   >2024      * Fuer X-Basic
VMBR    EQU   >202C      * Fuer X-Basic


MYWS    BSS   >20        * Workspace reservieren

BUFFER  BSS   >300       * Speicher fuer Bildschirm


VSAVE   LWPI  MYWS
        CLR   R0         * Start VDP-RAM bei >0000
        LI    R1,BUFFER
        LI    R2,>0300
        BLWP  @VMBR

ENDE    CLR   4
        MOVB  4,@>837C   * GPL-Statusbyte loeschen
        LWPI  >83E0
        B     @>0070     * Ruecksprung

VLOAD   LWPI  MYWS
        CLR   R0
        LI    R1,BUFFER
        LI    R2,>0300
        BLWP  @VMBW
        JMP   ENDE
```

Auch die Routinen VMBW und VMBR hatten wir schon angesprochen, so daß hier nur noch einmal kurz darauf eingegangen werden muß. Bei beiden Routinen muß R0 wieder die VDP-Adresse enthalten, R1 enthält einen Zeiger auf den Beginn des normalen RAM's, von wo aus geschrieben bzw. wohin gelesen werden soll und R2 enthält die Anzahl der Bytes. Das Listing 3 zeigt die Anwendung zum Schreiben eines Textes jeweils für das Assembler bzw. Mini-Memory Modul, wenn die Routine aus dem Basic bzw. Extended Basic aufgerufen wird und ein Beispiel dafür, wie der komplette Bildschirm in das RAM gespeichert wird.
Anzumerken zu VMBW und VMBR bleibt noch, daß niemals in R2 Null stehen darf, sonst hängt sich der Rechner bei einigen der Module auf. Hier ist wohl Texas Instruments eine kleine „Ungenauigkeit" unterlaufen. Normalerweise müßte dieser „Fehler" wohl in der Routine berücksichtigt werden.
Kommen wir nun zu einer weiteren wichtigen Hilfsroutine: KSCAN. Diese dient der Tastaturabfrage. Benötigt werden hier keine besonderen Vorbereitungen in den Registern unserer Workspace, sondern es müssen einige Speicherstellen gesetzt werden. Auf >8374 müssen wir ein Byte mit dem Wert des Tastaturmodus belegen. Dabei gelten die gleichen Werte wie beim Basic-Befehl CALL KEY. Bei dem Wert >01 wird also die linke Tastatur mit Joystick und bei >02 die rechte abgefragt. Die anderen Werte,> 00, >03, >04 und >05 sind identisch zum CALL KEY. Nach der Tastaturabfrage wird auf >8375 der ASCII-Wert der gedrückten Taste (auch hier gelten die gleichen Werte wie bei CALL KEY) und auf >8376 und >8377 gegebenenfalls die Werte für die Joysticks. Wenn eine neue Taste gedrückt wurde, ist das Bit 2 im GPL-Statusbyte (zu finden auf >837C) gesetzt, andernfalls ist das Bit (nicht das ganze Byte!) 0. Das Listing 4 zeigt zwei Anwendungen: Einmal wird nur gewartet, bis irgendeine Taste gedrückt wird und zum anderen wird solange gewartet, bis die Leertaste gedrückt wird.

```
* Listing 4:
       REF   KSCAN    * Fuer Ass. u. MM

KSCAN  EQU   >201C    * Fuer X-Basic
* Warten auf gedrueckte Taste:

       CLR   R4
       MOVB  R4,@>8374 * Tastaturmodus
       MOVB  R4,@>837C * GPL-Status loeschen
TASTE  BLWP  @KSCAN
       MOVB  @>837C,R4
       JEQ   TASTE

* Warten auf Leertaste

SPACE  DATA  >2000
       CLR   R4
       MOVB  R4,@>8374 * Tastaturmodus
TASTE  BLWP  @KSCAN
       MOVB  @>837C,R4
       COC   @SPACE,R4 * Bit 2 pruefen
       JNE   TASTE
       MOVB  @>8375,R4
       C     R4,@SPACE
       JNE   TASTE
```

Die letzte Der Hilfsroutinen soll XMLLNK sein. Diese erlaubt uns, auf einfache Art und Weise auf Routinen des Betriebssystem zuzugreifen. Je nachdem welche Routine wir benutzen wollen, müssen wir entsprechende Speicherbereiche vorbereiten. Wir wollen hier am Beispiel von CIF (Convert Integer – Floiting Point, also der Umwandlung einer Integer-Zahl in eine Fließkommazahl) den Gebrauch von XMLLNK zeigen (Listing 5). Dabei wird ein ganzes Wort, d.h. zwei Bytes in die im TI 99/4A benutzte Darstellung der Fließkomma-Zahlen umgewandelt.

```
* Listing 5:
* Fuer Assembler und Mini-Memory:

        REF   XMLLNK


        LI    R1,>0001 * Dieser Wert soll
        MOV   R1,@>834A * umgewandelt werden
        BLWP  @XMLLNK
        DATA  >2300     * Fuer Mini-Mem >7200
                        * Fehler im Handbuch

* Fuer Extended Basic:
XMLLNK  EQU   >2018
        LI    R1,>0001 * Dieser Wert soll
        MOV   R1,@>834A * umgewandelt werden
        BLWP  @XMLLNK
        DATA  >0020
```

Anzumerken zu XMLLNK bleibt noch, daß eine Veröffentlichung in einer anderen Zeitschrift hier zu etwas Unsicherheit geführt hat. Auch das XMLLNK des Extended Basic Moduls arbeitet einwandfrei, nur müssen andere DATA-Werte eingesetzt werden, wie es das Beispiel zeigt. Die Werte, wie auch sonstige Equates für das Extended Basic-Modul, finden sich in der Anleitung zum Editor-Assembler.

DER SCHLÜSSEL ZUM BETRIEBSSYSTEM

Zuerst wollen wir uns mit GPLLNK und DSRLNK beschäftigen. Diese beiden Routinen fehlen beim Extended Basic Modul.
Mittlerweile gibt es aber einige Veröffentlichungen über diese Hilfen auch bei Extended Basic, so sind auch Beispiele dafür in unserem Assembler-Sonderheft enthalten. Wollen wir diese Routinen also in einem Assembler-Programm für Extended Basic nutzen, so müssen wir diese einfach an den Schluß des Programms (oder Anfang) anhängen und können sie dann genauso wie beim Assembler-Modul oder beim Mini-Memory aufrufen.
GPLLNK ist in seiner Wirkung und der Handhabung dem XMLLNK sehr ähnlich. Im Gegensatz zu XMLLNK, welches Routinen aus dem ROM aufruft, können mit GPLLNK Routinen, die in den GROM's der Konsole bzw. des eingesteckten Moduls enthalten sind, abgearbeitet werden. Hier stehen in der Konsole diverse mathematische Funktionen für Fließkommazahlen und Routinen zum Laden der Zeichensätze, sowie zur Ausgabe von Tönen zur Verfügung. Ein Beispiel für die Benutzung der Routinen zum Laden der Zeichensätze in den VDP hatten wir schon im Heft 9/85. Die Benutzung der im Betriebssystem vorhandenen Routinen für "Beep" und "Onk", die Töne für richtige und falsche Eingabe, finden Sie in Listing 1.

```
* Listing 1:

        REF  GPLLNK * Fuer Assembler u. MM

* Fuer X-Basic muss GPLLNK angeh{ngt
* werden

* Accept-Tone:

        BLWP @GPLLNK
        DATA >0034

* Bad-Tone:

        BLWP @GPLLNK
        DATA >0036
```

Das geht doch wirklich einfach und kann in jedes Assembler-Programm eingebunden werden, ohne großes Programmieren von Soundlisten. Eigentlich ist es doch auch nicht einzusehen, warum nicht auch in Assembler-Programmen der Benutzer akustisch auf richtige und falsche Eingaben oder das Ende des Programms u.ä. hingewiesen wird. Bei den Ton-Routinen muß hier noch angemerkt werden, daß es einige Vorschläge für GPLLNK unter Extended Basic gibt, die hier nur einmal richtig funktionieren. Dazu gehört auch das Listing, welches TI/USA auf Anforderung verschickte. Die in unserem Assembler-Sonderheft abgedruckte Version arbeitet einwandfrei.
Kommen wir nun zum DSRLNK, und hier wird es etwas komplizierter. DSRLNK in seiner ursprünglichen Form, wie es im Betriebssystem des TI 99/4A enthalten ist, ist wohl der eigentliche Schlüssel des gesamten Betriebssystems. Darüber werden alle Haupt-Programme, Unterprogramme (auch die Basic-CALL's) und die Betriebssysteme der Peripheriegeräte wie z.B. Disk-Controller aufgerufen. Das DSRLNK für Assembler ist dagegen etwas abgemagert. Es erlaubt nur den Zugriff auf die Peripheriegeräte. Damit kann über diese Routine nicht auf den Kassettenrekorder zugegriffen werden. Das muß über GPLLNK erfolgen, wie es im Editor-Assembler-Handbuch und auch schon in der TI-REVUE kurz beschrieben wurde.
Um nun ein DSRLNK ausführen zu können, benötigen wir zuerst ein sogenannten PAB (Peripheral Access Block, was soviel heißt wie Zugriffs-Block für die Peripheriegeräte). Dieser muß wie folgt aufgebaut werden:
1. Byte: I/O Opcode (legt die Funktion fest) im Einzelnen:

> 01 OPEN
> 02 CLOSE
> 03 READ
> 04 WRITE
> 04 RESTORE
> 05 LOAD
> 06 SAVE
> 07 DELETE
> 08 SCRATCH RECORD
> 09 STATUS

OPEN, CLOSE, RESTORE, SAVE und DELETE haben die gleichen Bedeutungen wie in der Dateibehandlung im Basic, READ heißt Lesen eines Datensatzes, WRITE schreiben eines Datensatzes, LOAD ist gleichbedeutend mit OLD im Basic, SCRATCH RECORD bedeutet, daß der letzte Datensatz gelöscht werden soll und STATUS gibt Auskunft über die Form der Datei und ob irgendein Ende erreicht ist. Dies wird im Basic z.B. für die EOF-Funktion verwendet.
2. Byte: Flagbyte, enthält die wichtigen Informationen über den Typ der Datei.
Bit 0 (niedrigwertigstes Bit): Dateityp (1=Sequentiell, 0=Fixed)
Bit 1 und 2: Art der Eröffnung (00=Update, 01=Output, 10=Input und 11=Append)
Bit 3: Art der Daten (0=Display, 1=Internal)
Bit 4: Art des Datensatzes (=Fixed, 1=Variable)
Bit 5 bis 7: Errorcode, werden von der DSR entsprechend gesetzt, wenn ein Fehler bei der Dateibehandlung auftrat.
Byte 3 und 4: Zeiger zur Adresse des Puffers im VDP-RAM für den Datensatz.
Byte 5: Länge des Datensatzes. Bei Fixed-Dateien steht hier die Länge, bei Variable-Dateien die maximale Länge.
Byte 6: Länge des aktuellen Datensatzes, d.h. die Länge des gerade zu schreibenden oder gelesenen Datensatzes.
Byte 7 und 8: Nummer des Datensatzes. Bei SAVE steht hier die Länge des zu speichernden Programms,

```
* Listing 2:

* PAB fuer Display-Variable 80 Datei

PABVAR BYTE >00  * OPEN
       BYTE >14  * DISPLAY-VARIABLE, INPUT
       DATA >1000 * ADRESSE BUFFER
       BYTE >50  * MAX. 80
       BYTE >50  * MAX CHARACTER COUNT
       DATA >0000 * 1. DATENSATZ
       BYTE >00  * SCREEN-OFFSET 0
       BYTE >09
       TEXT 'DSK1.TEST' * Immer diese intelligenten Namen

       EVEN

RAMBUF BSS  80  Buffer fuer den Datensatz


* Hier vorher eigenes Programm

* PAB in das VDP-RAM legen:

       LI   R0,>0900      * Willkuerliche Adresse
       LI   R1,PABVAR
       LI   R2,19         * Komplette Laenge
       BLWP @VMBW

       MOV  R0,R3         * Sichern
       AI   R3,>0009      * Zeigt nun auf Laengenbyte
       BL   @DSRAUF
       JEQ  ERROR

* Nun ersten Datensatz lesen

       LI   R0,>0900
       LI   R1,>0200      * OPCODE fuer READ
       BLWP @VSBW         * In PAB schreiben

       BL   @DSRAUF       * Datensatz lesen
       JEQ  ERROR

       LI   R0,>0905      * Zeigt auf CHARACTER COUNT
       BLWP @VSBR         * Byte lesen
       MOVB R1,R2
       SRL  R2,8          * Laenge nun in R2
       JEQ  HILF          * Bei 0 Fehlfunktion VMBR
       LI   R0,>1000      * Buffer im VDP
       LI   R1,RAMBUF     * Buffer im CPU-RAM
       BLWP @VMBR         * Datensatz nun im Buffer CPU-Ram

* Nun Datei schliessen

HILF   LI   R1,>0100      * OPCODE fuer CLOSE
       LI   R0,>0900
       BLWP @VSBW         * In den PAB schreiben
       BL   @DSRAUF       * Datei schliessen
```

```
* Das wars, das eigene Programm kann weitergehen

* Hier Errorhandling einfuegen, In RØ Errorbyte
* bei Ø DSR nicht gefunden

ERROR  B    @BEGINN


* Unterprogramm Aufruf DSR

DSRAUF MOV  R3,@>8356
       BLWP @DSRLNK
       DATA >ØØØ8
       RT                  * Veraendert nicht Status
```

bei LOAD die max. Länge des Eingabebuffers, in den das Programm im VDP-RAM geladen werden soll. Ist das Programm länger, erfolgt eine Fehlermeldung.
Byte 9: Screen Offset: Wird wie im Basic für die Bildschirmdarstellung ein Screen-Offset verwendet, muß hier der Wert stehen.
Byte 10: Länge des Dateinamens. Dabei ist die gesamte Länge gemeint, also nicht nur die Länge des Namens des Peripheriegerätes.
Byte 11 folgende: Name
Diesen PAB müssen wir nun in das VDP-RAM bringen, und dann muß noch ein Zeiger auf >8356 (ganzes Wort) gelegt werden, der auf das Längenbyte des Namens im VDP-RAM zeigt. Danach kann mit dem üblichen BLWP mit nachfolgendem DATA das Betriebssystem eines Peripheriegerätes aufgerufen werden. Das folgende Listing 2 zeigt ein Beispiel dafür. Bitte beachten Sie, daß hier willkürlich in die Belegung des VDP-RAM's eingegriffen wird. Wenn Basic-Programme vorhanden sind, muß erst der entsprechende Speicherplatz für PAB und Datenbuffer reserviert werden. Dazu kommen wir ein anderes Mal.
Wenn nun ein Fehler auftritt, so ist bei der Rückkehr aus DSRLNK zum rufenden Programm das Equal-Bit im Statusregister der CPU gesetzt, d.h. durch einen einfachen Sprung, wenn gleich (JEQ), kann zu einer Routine gesprungen werden, die dann den Fehler entsprechend dem Programm behandelt.
Der letzte Hinweis für heute gilt der Eigentümlichkeit mancher Peripheriegeräte: Sie verändern die GROM-Adresse. Deshalb sollte vor dem Ausführen von DSRLNK die GROM-Adresse gelesen und danach wieder geschrieben werden. Wie das geht, zeigt das Listing 3.

```
* Listing 3:

* Grom-Adresse sichern vor DSRLNK und
* anschliessend wieder schreiben


SAVEGR DATA >ØØØØ

       MOVB @>98Ø2,@SAVEGR
       MOVB @>98Ø2,@SAVEGR+1
       DEC  @SAVEGR * Muss erniedriegt werden




       BLWP @DSRLNK
       DATA >ØØØ8



       MOVB @SAVGR,@>9CØ2
       MOVB @SAVEGR+1,@>9CØ2










‡U
)UUUUUUUUUU
```

PIXELWOLF

Mit dem Programm Pixelwolf kann in Extended Basic hochauflösende Grafik erzeugt werden. An Hardware benötigt man außer der Konsole das Extended Basic und die Mini-Assembler-Hardware-Erweiterung. Das Programm enthält einen Loader für das Maschinenprogramm. (Anm. d. Red.: Alle für den Mini-Assembler geschriebenen Programme lassen sich auch mit einer normalen Speichererweiterung betreiben.)

Programmbeschreibung

Die Zeilen 240 bis 300 regeln den Programmeinstieg. Hier wird entschieden, ob zunächst das Maschinenprogramm geladen wird (Menü-Punkt 1), oder ob sofort die Demonstrationsgrafik gestartet wird (Menü-Punkt 2). Zeile 340 setzt den Zeiger auf den Beginn der Namensliste für Maschinensprache-Programme und Zeile 350 trägt Programmname und Startadresse in diese Liste ein.
Die Zeilen 360–510 "Poken" das Maschinenprogramm in das Mini-Assembler-Ram. Ein mit einem Disassembler-Programm erstelltes Quellcode-Listing des Maschinenprogramms befindet sich in der Anlage. Das Maschinenprogramm belegt die Speicherplätze 24FE257F(hexadezimal) bzw. 9470–9599(dezimal).

Die Zeilen 520–540 initialisieren den Arbeitsbereich des Maschinenprogramms (2580–259F hexadezimal bzw. 9600–9631 dezimal) mit Null. Das Demo-Programm setzt dann in der Zeile 580 zunächst alle Zeichenmuster auf Leer. Zeile 600 lädt das Register 8 des Maschinenprogramms mit dem Wert dez. 128 entsprechend hex. 80, also dem Basic-Wert für das Leerzeichen.

Die Zeilen 610–710 plotten dann eine waagerechte und eine schräge Gerade sowie eine spezielle Sinuskurve auf den Bildschirm. Sie rufen dazu das Maschinenprogramm nicht direkt auf, weil die Parameterübergabe des Call-Link-Befehls in Verbindung mit dem Miniassembler-RAM nicht funktioniert. (Der Call-Link-Befehl des Extended-Basic sucht die numerischen Variablen vergeblich in der RAM-Erweiterung, während sie sich nach wie vor im VDP-RAM befinden.)

Parameterprüfung und -übergabe sowie der Maschinenprogrammaufruf werden durch eine Basic-Subroutine durchgeführt, die man in den Zeilen 760–790 findet.

Die Parameter werden direkt in das Register 0 des Maschinenprogramms geladen. Die Behandlung der Parameter und der Maschinenprogrammaufruf in der Basic-Subroutine haben den Vorteil, daß das Hauptprogramm entsprechend einfacher gestaltet werden kann.
Registerbelegung im Maschinen-Unterprogramm

```
100 ! PIXELWOLF
110 !
120 ! Hochaufloesende Grafik
130 ! fuer TI99/4A
140 ! mit EXTENDED BASIC
150 ! und MINIASSEMBLER
160 !
170 ! von Wolfgang Schmitt
180 !      Uhlandstr. 10
190 !      6238 Hofheim/Ts.
200 !
210 ! Menu
220 !
230 CALL CLEAR :: ON WARNING
 NEXT
240 DISPLAY AT(10,2):"1. Mas
chinenprogramm laden"
250 DISPLAY AT(12,2):"2. Dem
o-Programm starten"
260 DISPLAY AT(16,2):"Ihre W
ahl:"
270 ACCEPT BEEP SIZE(1)VALID
ATE("12")AT(16,27):A
280 IF A=2 THEN 580
290 DISPLAY AT(20,2):"Ladevo
rgang laeuft"
300 DISPLAY AT(22,2):"Bitte
warten"
310 !
320 ! Basic-Loader
330 !
340 CALL LOAD(8196,39,248)!
Beginn der Namensliste
350 CALL LOAD(10232,80,85,78
,75,84,32,36,254)! Programmn
ame und Startadresse
360 FOR I=9470 TO 9599
370 READ P :: CALL LOAD(I,P)
380 NEXT I ! Programm laden
390 DATA 2,224,37,128,4,193,
4,194,4,195
400 DATA 4,196,4,197,4,199,2
08,192,6,195
410 DATA 6,192,209,64,6,197,
16,0,16,0
420 DATA 2,12,0,8,60,140,61,
12,193,132
430 DATA 2,12,0,32,57,140,16
1,194,192,7
440 DATA 4,32,32,40,6,193,2,
129,0,128
450 DATA 22,7,5,136,194,72,1
92,72,6,193
460 DATA 4,32,32,32,16,1,194
,65,194,137
470 DATA 2,12;0,8,58,140,162
,197,192,11
480 DATA 4,193,4,32,32,40,6,
193,6,12
490 DATA 96,204,7,67,2,12,0,
1,192,3
500 DATA 19,1,10,12,224,76,6
,193,192,11
510 DATA 4,32,32,32,2,224,13
1,224,4,91
520 FOR I=9600 TO 9632
530 CALL LOAD(I,0)
540 NEXT I ! Arbeitsbereich
 initialisieren
550 !
560 ! Demo-Programm
570 !
580 FOR I=143 TO 33 STEP -1
:: CALL CHAR(I,"00000000000
0000"):: NEXT I
590 CALL CLEAR
600 CALL LOAD(9616,0,128)! Z
eichennamenzaehler auf Basic
-Wert fuer Leerzeichen
610 FOR X=10 TO 245 STEP 2
620 CALL PIWO(X,100)
630 NEXT X
640 FOR X=25 TO 225
650 Y=X/2
660 CALL PIWO(X,Y)
670 NEXT X
680 FOR X=20 TO 235 STEP .2
690 Y=SIN(X/20)*(100-X)+100.
5
700 CALL PIWO(X,Y)
710 NEXT X
720 GOTO 720
730 !
740 ! Routine fuer Parameter
-Uebergabe und Maschienenpro
gramm-Aufruf
750 !
760 SUB PIWO(X,Y)
770 IF X<1 OR X>256 OR Y<1 O
R Y>192 THEN 790
780 CALL LOAD(9600,X,Y):: CA
LL LINK("PUNKT")
790 SUBEND
```

R0	Parameterübergabe oder VDP-RAM-Adresse
R1	Von/nach VDP-RAM zu übertragendes Byte
R2	X-grob
R3	X bzw. X-fein
R4	Y-grob
R5	Y bzw. Y-fein
R6/7	Bildschirmadresse
R8	Zeichennamen-Zähler
R9	aktueller Zeichenname
R10/11	Byte-Adresse in der Zeichendefinitionsliste
R12	Bit-Muster für neuen Pixel
R13–15	nicht benutzt

Bedienung

1. Mit "FUNCTION QUIT" das Titelbild aufrufen und Extended-Basic anwählen.
2. Miniassembler-RAM einschalten.
3. Durch Eingabe von "CALL INIT <ENTER> NEW <ENTER>" werden diverse Hilfsprogramme von Extended-Basic-Modul in das Miniassembler-RAM geladen.
4. Mit "OLD CS1" das Programm Pixelwolf laden und mit "RUN" starten.
5. Beim ersten Programmstart muß aus dem Menü-Punkt 1 (Maschinenprogramm laden) angewählt werden. Später kann dann mit Punkt 2 direkt die Demografik aufgerufen werden.
6. Das Programm beendet man mit "FUNCTION CLEAR".

Wolfgang Schmitt

PALINDROM

Ein Palindrom ist eine Zahl, die von vorwärts und rückwärts gelesen den gleichen Wert darstellt, z.B. 2992. Man kann aus fast allen Zahlen durch bestimmte Additionen ein Palindrom erzeugen. Man beginnt mit einer beliebigen zwei- oder mehrstelligen Zahl, schreibt die gleiche Zahl in umgekehrter Ziffernfolge darunter und addiert beide Zahlen. Mit der Summe wird wieder so verfahren. Diese Reihe wird solange fortgesetzt, bis ein Palindrom erreicht wird. Das folgende Beispiel dient zur Illustration:

```
  194
+ 491
-----
  685
+ 586
-----
 1271
+1721
-----
 2992 Palindrom
```

Die meisten Ausgangszahlen führen sehr schnell zu einem Palindrom. Es gibt aber zwischen 100 und 200 eine Zahl, die anscheinend zu keinem Palindrom führt. Ich überlasse es dem Leser, diese Zahl ausfindig zu machen; per Hand oder mit Hilfe des Assembler-Programmes. Soweit mir bekannt ist, erreicht diese Ausgangszahl selbst nach 50 000 Additionen (das führt zu einer Zahl mit über 20 000 Stellen) noch keine Palindromform.

Zum Programm

Um das Programm nicht unnötig zu verlängern, wurde auf Dialoge verzichtet. Deshalb sind einige Erklärungen notwendig: Nach Eingabe einer Anfangszahl und 'ENTER' werden die errechneten Summen angezeigt, bis der Bildschirm gefüllt oder ein Palindrom erreicht ist. Auf Tastendruck wird die Rechnung fortgesetzt, bzw. zur Eingabe einer neuen Anfangszahl aufgefordert. Mit 'Quit' wird das Programm verlassen. Sollte die Summenzahl 28 Stellen erreicht haben ohne daß ein Palindrom erreicht wurde, wird ebenfalls nach Tastendruck zur Eingabe einer neuen Anfangszahl aufgefordert.

Der Debugger

Der Autor gehört zu den Computerfans, die versucht haben, ohne jede Vorkenntnis die Assemblerprogrammierung mit Hilfe des TI-Handbuches zu erlernen. Es war am Anfang ausgesprochen frustrierend, aber Ausdauer führte zu einem ersten Resultat.
Da Assemblerprogramme die dumme Angewohnheit haben, am Anfang nie zu laufen und sich statt dessen irgendwo aufzuhängen, gehört der Debugger zu den wichtigsten Hilfsmitteln, um Fehler ausfindig zu machen.
Für diejenigen, die sich erst kurz mit dem Assembler beschäftigen oder ihn entnervt in einer Schublade aufbewahren, möchte ich anhand des Palindromprogrammes die wichtigsten Debuggerbefehle erläutern.
Nachdem Sie das Programm eingetippt haben, speichern Sie es unter DSK2.PS (Palindrom Source) ab. Bei der anschließenden Assemblierung geben Sie für das Objektfile DSK2.P0 und für das Listfile DSK2.PL ein und starten mit der Option RL. Nach erfolgter Assemblierung drukken Sie das PL-File aus. In der ersten Spalte sehen Sie die Zeilennummern und in der zweiten Spalte die relative Speicheradresse. Um zu der tatsächlichen Adresse zu kommen, müssen Sie jeweils >A000 addieren, da die LOAD und RUN Option das Object-File im High-Memory ab A000 abspeichern wird.
Wählen Sie jetzt die 'LOAD and RUN' Option und geben Sie folgendes ein: DSK2.P0, Enter, DSK1.DEBUG, Enter, Enter, DEBUG. Mit M A000,FFFF zeigt Ihnen der Rechner die Speicherbelegung ab Adresse A000. Durch Tastendruck können Sie die Anzeige unterbrechen und wieder starten, Fctn X bricht den Befehl ab. Viel werden Sie nicht erkennen, lediglich das Auftauchen des Wortes 'PALINDROM' zeigt Ihnen, daß es sich hier wirklich um Ihr Programm handelt.

Als nächstes wollen wir das Programm aus dem Debugger heraus starten. Dazu drücken wir 'R'.
Als Prompt wird der derzeitige Inhalt des Workspace Pointers W angezeigt, den wir durch Eingabe von A078 auf die Startadresse unseres selbst definierten Workspace MYREG (siehe PL-Ausdruck) einstellen und durch Drücken der Leertaste zum Programm Counter P weiterschalten. Die angezeigte Adresse ändern wir nach A0E6, der Startadresse unseres Programms. Durch Drükken der Leertaste kommen wir zum Statusregister, das wir auf 0000 abändern, falls dieser Wert nicht schon angezeigt wird und beenden diesen Abschnitt mit Tastendruck.

Als nächstes setzen wir einen Breakpoint durch Eingabe vom B A10C und starten unser Programm mit E. An dem vorgewählten Breakpoint hält das Programm an und wir können mit M A000,FFFF die Auswirkung des ersten Programmabschnitts betrachten. Sie werden feststellen, daß die Blöcke für Z1, Z2 und Buffer mit Nullen bzw. mit dem ASCII-Code für Leerstellen gefüllt sind. Nach Eingabe von W können auch die Inhalte der Register R0 bis R15 inspiziert werden.

Sie können jetzt einen oder mehrere Breakpoints setzen und mit dem E Befehl abschnittsweise durch das Programm laufen.

```
       DEF  START
       REF  KSCAN,VMBW,VSBW,VMBR
*
H1     BYTE 1                     H1=1, Byte-Konstante
H1Ø    BYTE 1Ø                    H1Ø=1Ø
H4B    BYTE 48                    H4B=48, ASCII-Code fuer Null
       EVEN
Z1     BSS  28                    Block von 28 Byte, zu untersuchende Zahl
Z2     BSS  28                    Hilfsblock fuer Addition
BUFFER BSS  28                    Block fuer Z1 im ASCII-Code
BUF1   BSS  32                    Buffer fuer Bildschirm-Scroll
MYREG  BSS  32                    Block fuer Arbeitsregister
TITEL  TEXT '*************'       Text fuer Programmtitel
       TEXT '*           *'
       TEXT '* PALINDROM *'
       TEXT '*           *'
       TEXT '*************'
       EVEN
FRAGE  TEXT 'ANFANGSZAHL?'        Text fuer die Eingabe der Anfangszahl
*
START  LWPI MYREG                 Arbeitsregister laden
*
* Titelbild
*
       LI   RØ,>48                RØ=Adresse der 3. Bildschirmzeile, 9. Spalte
       LI   R1,TITEL-13           R1=Adresse 13 Byte vor Titel
       LI   R2,13                 R2=13, Anzahl der zu schreibenden Bytes
LOOP3  AI   RØ,32                 RØ=RØ+32, d.h. eine Bildschirmzeile tiefer
       AI   R1,13                 R1=R1+13, d.h. naechste Textzeile
       BLWP @VMBW                 Textzeile auf dem Bildschirm anzeigen
       CI   RØ,>E8                RØ=>E8 ? (Bildschirmposition der 5. Textzeile)
       JNE  LOOP3                 Nein -> naechste Textzeile  anzeigen
*
NEU    BL   @INIT                 Zahlen initialisieren
*
* Anfangszahl eingeben
*
       LI   RØ,>2A2               22. Bildschirmzeile, 3. Spalte
       LI   R1,FRAGE              R1 mit der Adresse von 'FRAGE' laden
       LI   R2,12                 12 Bytes sollen uebertragen werden
       LIMI Ø                     Interrupt unterdruecken
       BLWP @VMBW                 Textzeile auf dem Bildschirm anzeigen
       LIMI 2                     Interrupt zulassen
*
       LI   RØ,>2BØ               22. Bildschirmzeile, 17. Spalte
NEXT8  BL   @SCAN                 Unterprogramm SCAN aufrufen
       CI   R1,>ØDØØ              wurde ENTER (ASCII-Code >ØD) gedrueckt?
       JEQ  NEXT9                 Ja -> weiter mit NEXT9
       CI   R1,>3ØØØ              ASCII-Code der gedrueckten Taste < ASCII Null ?
       JLT  NEXT8                 Ja -> Annahme verweigern, noch einmal
       CI   R1,>39ØØ              ASCII-Code der gedrueckten Taste > ASCII 9 ?
       JGT  NEXT8                 Ja -> Annahme verweigern, noch einmal
       INC  RØ                    Spaltenadresse erhoehen
       LIMI Ø                     Interrupt unterdruecken
       BLWP @VSBW                 gedrueckte Taste auf dem Bildschirm anzeigen
       LIMI 2                     Interrupt zulassen
       JMP  NEXT8                 naechste Ziffer holen
*
* Zahl vom Bildschirm nach Z1 uebertragen
*
NEXT9  LI   R2,>2BØ               ein Byte vor der ersten Ziffer
       NEG  2                     R2=-R2
       A    RØ,R2                 R2=RØ+R2, d.h. R2=Anzahl der eingegebenen Ziffern
       LI   RØ,>2B1               ab hier soll vom Bildschirm gelesen werden
       LI   R1,Z1+28              R1=Adresse von Z1+28
       S    R2,R1                 R1=R1-R2, d.h. R1=Adresse der 1. Ziffer in Z1
       LIMI Ø                     Interrupt unterdruecken
       BLWP @VMBR                 eingegebene Zahl vom Bildschirm nach Z1 kopieren
       LIMI 2                     Interrupt zulassen
       BL   @ZAHL                 Unterprogramm ZAHL aufrufen
*
* Pointer auf 1. Ziffer von Z1
*
LOOP2  LI   R4,-1                 R4=-1
LOOP1  INC  R4                    R4=R4+1
       MOVB @Z1(R4),@Z1(R4)       Z1 byteweise auf sich selbst kopieren, dient nur
       JEQ  LOOP1                 der Abfrage ob Ziffer von Z1 noch null
       CI   R4,Ø                  Pointer = Ø, d.h. Z1 ist 28 Ziffer lang ?
       JEQ  STOP                  ja -> Stop
*
* Aufruf von Unterprogrammen
*
NEXT6  BL   @STRING
       BL   @SHOW
       BL   @PALIN
*
* Bildschirm einmal gefuellt ?
*
       DEC  R6                    R6=R6-1
       JNE  NEXT                  R6=Ø? Nein -> noch keine 24 Zahlen angezeigt
       BL   @SCAN                 Bildschirm mit Zahlen voll, Halt, auf Tasten-
       LI   R6,24                 druck warten, dann R6 wieder gleich 24 setzen
*
* Z1 nach Z2 kopieren
*
NEXT   LI   R2,28                 R2=28
NEXT1  DEC  R2                    R2=R2-1
       MOVB @Z1(R2),@Z2(R2)       die Zahl wird byteweise kopiert
       C    R2,R4                 sind schon alle Ziffern kopiert?
       JNE  NEXT1                 Nein -> NEXT1
*
* Addition Z1 + Z2 =>Z1
*
       LI   R2,28                 R2=28, max. Anzahl von Ziffern in Z1
       MOV  R4,R3                 Pointer nach R3 kopieren
       DEC  R3                    R3=R3-1
NEXT2  DEC  R2                    R2=R2-1
       INC  R3                    R3=R3+1
       CLR  R5                    R5=Ø
       AB   @Z2(R3),@Z1(R2)       Z2 von vorne, Z1 von hinten ziffernweise addieren
       MOVB @Z1(R2),R5            Ziffernsumme nach R5 kopieren
       SWPB 5                     und ins LSB bringen
       CI   R5,9                  groesser 9 ?
       JGT  NEXT3                 ja -> NEXT9, Uebertrag durchfuehren
       C    R4,R2                 schon alle Ziffern addiert ?
       JNE  NEXT2                 nein -> naechste Addition durchfuehren
       JMP  LOOP2                 ja -> LOOP2, Pointer neu einstellen
*
```

```
NEXT3   SB    @H10,@Z1(R2)      von der Summenziffer 10 subtrahieren
        AB    @H1,@Z1-1(R2)     zu der Ziffer davor 1 addieren
        C     R2,R4             schon alle Ziffern addiert ?
        JNE   NEXT2             nein -> naechste Addition durchfuehren
        JMP   LOOP2             ja -> LOOP2, Pointer neu einstellen
*
* naechste Anfangszahl
*
STOP    BL    @SCAN             Tastatur abfragen, falls Taste gedrueckt weiter
        LI    R1,>2000          MSB von R1 mit ASCII-Code fuer Leertaste laden
        LI    R0,>2FF           R0 = Adresse der unteren rechten Bildschirmecke
        LIMI  0                 Interrupt unterdruecken
LOOP5   BLWP  @VSBW             auf dem Bildschirm anzeigen
        DEC   R0                Bildschirmadresse erniedrigen
        JNE   LOOP5             Bildschirm vollstaendig geloescht ? nein -> LOOP5
        LIMI  2                 Interrupt zulassen
        JMP   NEU               naechste Anfangszahl holen
*
* Beginn der Unterprogramme
* =======================
*
* Abfrage ob Palindrom erreicht
*
PALIN   LI    R2,28             max. Anzahl der Ziffern in Z1
        MOV   R4,R3             Pointer nach R3 kopieren
        DEC   R3                R3=R3-1
NEXT4   DEC   R2                R2=R2-1
        INC   R3                R3=R3+1
        CB    @Z1(R2),@Z1(R3)   Ziffern von Z1 gegenlaeufig vergleichen
        JEQ   NEXT5             Ziffern sind gleich ?  ja -> NEXT5
        RT                      zurueck ins Hauptprogramm
NEXT5   C     R2,R4             Alle Ziffern verglichen ?
        JEQ   STOP              ja -> STOP
        JMP   NEXT4             nein -> naechstes Ziffernpaar vergleichen
*
* Z1 auf den Bildschirm bringen
*
SHOW    LI    R2,32             R2=32, Anzahl der Positionen pro Zeile
        LI    R0,-32            R0=-32
        LI    R1,BUF1           R1 enthaelt Adresse des RAM-Buffers
        LIMI  0                 Interrupt unterdruecken
SC      AI    R0,64             Leseadresse um 64 Byte (2 Zeilen) erhoehen
        BLWP  @VMBR             32 Byte vom Schirm in den RAM-Buffer kopieren
        S     R2,R0             Schreibadresse um 32 Byte (1 Zeile) erniedrigen
        BLWP  @VMBW             32 Byte aus dem RAM-Buffer ins VDP-RAM kopieren
        CI    R0,>02C0          letzte Zeile kopiert ?
        JNE   SC                nein -> weiter scrollen
*
        LI    R0,>02E2          Schreibadresse = unterste Zeile, 3. Spalte
        LI    R1,BUFFER         Leseadresse = Buffer, enthaelt ASCII-Code von Z1
        LI    R2,28             28 Byte
        BLWP  @VMBW             ins VDP-RAM uebertragen
        LIMI  2                 Interrupt zulassen
        RT                      zurueck ins Hauptprogramm
*
* Zahl in String unwandeln
*
STRING  LI    R1,28             max. 28 Byte sind zu transformieren
NEXT7   DEC   R1                R1=R1-1
        MOVB  @Z1(R1),@BUFFER(R1)   Z1 nach Buffer kopieren
        AB    @H48,@BUFFER(R1)  zu jeder Ziffer den ASCII-Code von Null addieren
        C     R1,R4             alle Ziffern transformiert ?
        JNE   NEXT7             nein -> naechste Ziffer
        RT                      zurueck ins Hauptprogramm
*
* String in Zahl umwandeln
*
ZAHL    SB    @H48,*R1          ASCII-Code von Null subtrahieren
        INC   R1                R1=R1+1
        DEC   R2                R2=R2-1
        JGT   ZAHL              R2>0 ? ja ->  naechste Ziffer transformieren
        RT                      zurueck ins Hauptprogramm
*
* Tastatur scannen
*
SCAN    LI    R3,>2000          R3 = 0010000000000000
        CLR   R1                R1=0
        LIMI  0                 Interrupt unterdruecken
        BLWP  @KSCAN            Tastatur scannen
        LIMI  2                 Interrupt zulassen
        MOV   @>837C,R2         Statusbyte nach R2 kopieren
        COC   R3,R2             Einsen vergleichen, d.h. wurde Taste gedrueckt ?
        JNE   SCAN              nein -> noch einmal scannen
        MOVB  @>8375,R1         ja -> ASCII-Code der Taste nach R1 kopieren
        RT                      zurueck ins Hauptprogramm
*
* Z1,Z2 mit Nullen, Buffer mit Leerzeichen fuellen
*
INIT    LI    R6,24             Zaehler fuer einen kompletten Bildschirm-Scroll
        CLR   R1                R1=0
        LI    R2,>2020          R2 mit ASCII-Code fuer Leerzeichen laden
        LI    R3,28             28 Bytes sollen uebertragen werden
LOOP4   MOV   R1,@Z1-2(R3)      Z1 von oben mit Nullen fuellen
        MOV   R1,@Z2-2(R3)      Z2 "    "    "    "
        MOV   R2,@BUFFER-2(R3)  Buffer mit Leerzeichen fuellen
        DECT  R3                R3 um 2 erniedrigen
        JNE   LOOP4             R3=0 ? Nein -> nach LOOP4 springen
        RT
*
        END
```

Ich hoffe, durch diese detaillierte Beschreibung allen Mut gemacht zu haben, die bis jetzt vor der Assemblerprogrammierung zurückgeschreckt sind oder frustriert wieder aufgegeben haben.

Dr. Uwe Schulze

UTILITIES FÜR INTERRUPT-GESTEUERTE SOUNDLISTEN

Dieses Programm erweitert Ihren TI 99/4A um folgende Maschinen-Routinen:

CALL LINK("SLINIT", Bytezahl) initialisiert die Soundlistenerstellung und muß vor allen anderen Sound-Routinen stehen. Es ist empfehlenswert, diesen Befehl folgendermaßen in einem LOAD-Programm unterzubringen:

```
100 CALL INIT
110 CALL LOAD
      ("DSK1.UTILITIES")
120 CALL LINK
      ("SLINIT",1000)
```

Die 1000 ist dabei die Anzahl der im VDP-RAM zu reservierenden Bytes.

CALL LINK ("LOESCH") reinitialisiert den Speicherbereich im VDP-RAM, der von den Soundlisten benutzt wird. Darauf muß ein NEW folgen.

CALL LINK ("LABEL", Label) definiert ein Label von maximal 6 Zeichen, daß von den Unterprogrammen "SPRUNG" und "XSOUND" angesprungen werden kann. Es können bis zu zehn Labels definiert werden. Bei einem Überlauf wird eine MEMORY FULL Fehlermeldung ausgegeben.

CALL LINK ("FREQ1", Frequenz) gibt die Frequenz für Tongenerator #1 an (Frequenz 110–32767 Hz).

CALL LINK (("FREQ2", Frequenz) gibt die Frequenz für Tongenerator #2 an (Frequenz 110–32767 Hz).

CALL LINK ("FREQ3", Frequenz) gibt die Frequenz für Tongenerator #3 an (Frequenz 110–32767 Hz).

CALL LINK ("FREQ4", Frequenz) gibt die Frequenz und Art (weißes Rauschen oder periodisches Rauschen) des Geräuschgenerators an (siehe Seite 86 der Bedienungsanleitung zum TI 99/4A).

CALL LINK ("LAUT1". Lautstärke) gibt die Lautstärke für Tongenerator #1 an (Lautstärke 0–30).

CALL LINK ("LAUT2", Lautstärke) gibt die Lautstärke für Tongenerator #2 an (Lautstärke 0–30).

CALL LINK ("LAUT3", Lautstärke) gibt die Lautstärke für Tongenerator #3 an (Lautstärke 0–30).

CALL LINK ("LAUT4", Lautstärke) gibt die Lautstärke des Geräuschgenerators an (Lautstärke 0–30).

CALL LINK ("DAUER". Dauer) definiert die Dauer des Tons, der zuvor durch die Unterprogramme "FREQ" und "LAUT" bestimmt wurde. Jede Frequenz- oder Lautstärkenangabe muß mit DAUER abgeschlossen werden (Dauer 0–255 in 1/50 Sekunden gemessen). Wenn als Dauer 0 angegeben wird, stoppt die interruptgesteuerte Soundlistenverarbeitung bis sie durch einen "XSOUND"-Befehl wieder neu begonnen wird.

CALL LINK ("SPRUNG", Label) Wird dieser Befehl vom Interruptprogramm gefunden, springt es zu einem vorher definiertem Label und verarbeitet die dort stehende Soundliste.

CALL LINK ("XSOUND", Label) startet die Soundlistenverarbeitung an der Stelle, an der das Label definiert wurde.

CALL LINK ("STOP") unterbricht die Soundlistenverarbeitung und löscht alle Ton- und Geräuschgeneratoren.

Mit CALL PEEK (−31794,X) kann man abfragen, ob das Ende der Soundliste bereits erreicht ist. Wenn X=0 ist, wird gerade keine Soundliste abgearbeitet. Zum besseren Verständnis der Befehle sind hier zwei kleine Demos abgedruckt. Dieses Beispiel erzeugt das Geräusch eines Schusses:

```
1 REM***********************
2 REM*                     *
3 REM*    SOUND-DEMO       *
4 REM*   ------------      *
5 REM*                     *
6 REM* 6.7.85 T. MIELKE    *
7 REM*                     *
8 REM***********************
9 REM
10 CALL LINK("SLSTOP")
11 ALL LINK("LOESCH")
12 ALL LINK("LABEL","SCHUSS")
13 CALL LINK("FREQ4",-8)
14 CALL LINK("LAUT4",0)
15 FOR I=0 TO 30
16 CALL LINK("FREQ3",4000-I*30)
17 CALL LINK("LAUT4",I)
18 CALL LINK("DAUER",1)
19 NEXT I
20 CALL LINK("LAUT4",30)
21 CALL LINK("DAUER",0)
22 CALL LINK("XSOUND","SCHUSS[illegible]
23 PRINT "BITTE TASTE DRUECKEN!": :
: : :
24 CALL KEY(0,A,B)
25 IF B<>1 THEN 24
26 CALL LINK("XSOUND","SCHUSS")
27 GOTO 24
```

Unter dem folgenden Geräusch könnte man sich den Start einer Rakete vorstellen:

Laden Sie den Object-Code, nachdem Sie den Speicher mit CALL INIT initialisiert haben. Geben Sie CALL LINK ("SLINIT",1000) und NEW ein. Danach prüfen Sie mit einer der Demos, ob das Programm funktioniert.

An dieser Stelle soll die Routine zur Reservierung von statischem Speicherplatz im VDP-RAM in Harmonie mit dem Basic-Interpreter ausführlich erklärt werden. Der Diskcontroler benutzt die Adresse >8370 (VDP-RAMTOP) als Zeiger zum Anfang der Diskbuffer. Was liegt näher, als den benötigten VDP-RAM-Buffer selbst als Disk-Buffer zu initialisieren? Zuerst werden zur Bufferlänge vier Bytes dazugezählt. Vom VDP-RAMTOP wird jetzt die Bufferlänge abgezogen. >8370 zeigt nun auf den Beginn Ihres Disbuffers. Dort müssen Sie nur noch den Header anfügen.

Er besteht aus vier Bytes:
1. Byte: >AA Erkennungsbyte für den Disk-Controler
2. und 3. Byte: Zeiger auf den nächsten Diskbuffer
4. Byte: >00 CRU-Adresse (hier keine)

Nach diesen vier Bytes stehen Ihnen alle folgenden Bytes bis zum nächsten Diskbuffer zur Verfügung. Nach dem Aufruf dieser Routine (siehe SLINIT im Quellisting) durch CALL LINK und NEW wird der Speicherplatz im VDP-RAM für Ihre Zwecke reserviert. Ein CALL FILES sollte danach vermieden werden.

Thomas Mielke

```
*************************************************************************
*                                                                       *
*        Interruptgesteuerte Soundlistenverarbeitung fuer BASIC         *
*       ------------------------------------------------------         *
*                                                                       *
*     Fertiggestellt am 1Ø.Ø9.1985                 Thomas Mielke        *
*                                                  Falkenkamp 17b       *
*     Konfiguration: TI 99, X-BASIC, 32K, Disk     2ØØØ Norderstedt     *
*                                                                       *
*************************************************************************

       DEF  SLINIT,LABEL,FREQ1,LAUT1,FREQ2,LAUT2,FREQ3,LAUT3
       DEF  FREQ4,LAUT4,DAUER,SPRUNG,XSOUND,STOP,LOESCH

VSBW   EQU  >2Ø2Ø
VMBW   EQU  >2Ø24
XMLLNK EQU  >2Ø18
SOUND  EQU  >84ØØ
STRREF EQU  >2Ø14
NUMREF EQU  >2ØØC
ERR    EQU  >2Ø34
RAMTOP EQU  >837Ø

LAENGE DATA Ø              * Adresse des Laengenbytes am Anfang des Eintrags
SBYTE  DATA Ø              * aktuelles Sound-Byte (bei der Listenerstellung)
SCOUNT DATA Ø              * zur Berechnung der Laenge eines Listeneintrags
SBOUND DATA Ø              * Obere Grenze des Bereichs fuer Soundlisten
LCOUNT DATA Ø              * Anzahl der benutzten Marken
LBUFF  BSS  8Ø             * Buffer fuer 1Ø Labels
       BYTE Ø
LENGHT BYTE Ø              * Laengenbyte fuer Marke
BUFFER BSS  6              * Buffer fuer eine Sprungmarke
HØ1    BYTE >Ø1
       EVEN

* Routine vergleicht aktuelles Sound-Byte mit der oberen Grenze
COMP
       C    @SBYTE,@SBOUND
       JL   COMP1
       DECT @SBYTE
       LI   RØ,>ØBØØ
       BLWP @ERR
COMP1  B    *R1Ø

* Routine liest ein numerisches Parameter in RØ ein
NUMBER
       CLR  RØ             * Null fuer einfache Variable
       LI   R1,1           * Parameternummer
       BLWP @NUMREF        * Fliesskommazahl holen
       BLWP @XMLLNK        * Rom-Routine
       DATA >12B8          * Fliesskommazahl in ganze Zahl umwandeln
       LI   RØ,>3ØØ        * >Ø3 = Ueberlauf-Fehlercode
       CB   RØ,@>8354      * Ueberlauf?
       JNE  NUMBE1         * ja, dann Sprung nach NUMBE1
       LI   RØ,>1EØØ       * Fehlercode BAD VALUE
       BLWP @ERR           * Fehler ausgeben und zurueck ins BASIC
NUMBE1 MOV  @>834A,RØ      * RØ mit Integerzahl laden
       RT                  * Ruecksprung

* Routine liest String in String-Buffer ein
STRING
       LI   RØ,>2Ø2Ø       * zwei Leerzeichen
       LI   R1,BUFFER      * Bufferadresse
       MOV  RØ,*R1+        *
       MOV  RØ,*R1+        *  Buffer mit Leerzeichen fuellen
       MOV  RØ,*R1+        *
       CLR  RØ             * einfache Stringvariable
       LI   R1,1           * Parameternummer
       LI   R2,>6ØØ        * max. Laenge laden
       MOVB R2,@LENGHT     * Laenge schreiben
       LI   R2,LENGHT      * Adresse des Laengenbytes
       BLWP @STRREF        * String holen
       RT

* Routine ermittelt Frequenzwert
DIVIDE
       LI   R1,1           * hoeherwertiges Wort von 11186Ø
       LI   R2,46324       * niederwertiges Wort von 11186Ø
       DIV  RØ,R1          * durch RØ teilen (Ergebnis in R1)
       RT

* Routine sucht Marke im Label-Buffer
MARKE
       LI   RØ,LBUFF       * RØ mit der Adresse des Buffers laden
MARKE1 MOV  RØ,R1          * RØ kopieren
       C    *RØ+,@BUFFER   *
       JNE  MARKE2         *
       C    *RØ+,@BUFFER+2 * Vergleich:
       JNE  MARKE2         * Wenn Label nicht gefunden, Sprung nach MARKE2
       C    *RØ+,@BUFFER+4 *
       JNE  MARKE2         *
       MOV  *RØ,RØ         * VDP-Adresse nach RØ laden
       RT                  * Ruecksprung
MARKE2 MOV  R1,RØ          * Altes RØ zurueckholen
       AI   RØ,8           * 8 dazu
       MOV  @LCOUNT,R2     * Anzahl der Eintraege im Label-Buffer
       AI   R7,LBUFF       * + Adresse des Label-Buffers
       C    RØ,R2          * Label-Buffer leer?
       JL   MARKE1         * Nein, dann zurueck nach MARKE1
       LI   RØ,>1EØØ       * Fehlercode BAD VALUE
       BLWP @ERR           * Fehlermeldung ausgeben und zurueck ins BASIC

* Pseudo-Diskbuffer fuer Soundlisten im VDP errichten und
* Soundlistengenerierung initialisieren
SLINIT
       MOV  R11,R1Ø        * Ruecksprung sichern
       BL   @NUMBER        * Anzahl der zu reservierenden Bytes in RØ
       AI   RØ,4           * 4 dazu
       MOV  @RAMTOP,R2     * Altes VDP-RAMTOP in R2 ablegen
       MOV  R2,@SBOUND     * Obere Grenze fuer Soundlisten
       DECT @SBOUND        * - 2
       S    RØ,@RAMTOP     * Anzahl der Bytes vom VDP-RAMTOP abziehen
       MOV  @RAMTOP,RØ     * Neues RAMTOP in RØ
       INC  RØ             * RØ = erstes Byte des neuen Bereichs im VDP-RAM
       LI   R1,>AAØØ       * Header fuer Disk-Buffer
       BLWP @VSBW          * schreiben
       INC  RØ             * + 1
       MOV  R2,R1          * High Byte des alten RAMTOP als Pointer zum
```

```
        BLWP @VSBW          * naechsten Disk-Buffer
        INC  RØ             * + 1
        SWPB R1             * Bytes tauschen
        BLWP @VSBW          * Low-Byte ins VDP-RAM schreiben
        INC  RØ             * + 1
        CLR  R1             * Ø da keine CRU-Adresse belegt wird
        BLWP @VSBW          * schreiben
        INC  RØ             * RØ = erstes freies Byte im VDP-Buffer
SLINØ1  MOV  RØ,@SBYTE      * aktuelles Soundbyte laden
        CLR  @LCOUNT        * Markenzaehler loeschen
        CLR  @SCOUNT        * Soundbyte-Zaehler loeschen
        MOV  RØ,@LAENGE     * Adresse des Laengenbytes fuer Eintrag laden
        INC  @SBYTE         * akt. Soundbyte + 1
        B    *R1Ø           * Ruecksprung

* Soundlistengenerierung initialisieren
LOESCH
        MOV  R11,R1Ø        * Ruecksprung sichern
        MOV  @RAMTOP,RØ     * RØ mit RAMTOP laden
        AI   RØ,5           * Header fuer Diskbuffer ueberspringen
        JMP  SLINØ1         * weiter in SLINIT

* Marke setzen
LABEL
        MOV  R11,R1Ø        * Ruecksprung sichern
        BL   @STRING        * String in String-Buffer einlesen
        LI   RØ,BØ          * Obere Grenze des Label-Buffers
        C    @LCOUNT,RØ     * Buffer voll?
        JL   LABEL1         * Nein, nach LABEL1 verzweigen
        LI   RØ,>ØBØØ       * Fehlercode MEMORY FULL
        BLWP @ERR           * Fehler ausgeben und zurueck ins BASIC
LABEL1  LI   RØ,BUFFER      * Adresse des eingelesenen Labels
        MOV  @LCOUNT,R1     * Labelzaehler in R1
        AI   R1,LBUFF       * + Adresse des Label-Buffers
        MOV  *RØ+,*R1+      *
        MOV  *RØ+,*R1+      * String aus dem String-Buffer in den Label-Buffer
        MOV  *RØ+,*R1+      *
        MOV  @SBYTE,*R1     * akt. Sound-Byte wird der Wert fuer das Label
        DEC  *R1            * Wert aufs Laengenbyte des Listeneintrags legen
        LI   RØ,B           * RØ = B
        A    RØ,@LCOUNT     * Markenzaehler erhoehen
        B    *R1Ø           * Ruecksprung

* Routine generiert Frequenzeintrag fuer Soundgenerator 1
FREQ1
        MOV  R11,R1Ø        * Ruecksprung retten
        BL   @NUMBER        * Zahl in RØ holen
        BL   @DIVIDE        * Frequenzwert ermitteln und in R1 ablegen
        MOV  R1,R2          * R1 sichern
        SLA  R1,12          * die vier niederwertigen Frequenz-Bits isolieren
        SRL  R1,4           *
        AI   R1,>BØØØ       * OP-Code dazu
        MOV  @SBYTE,RØ      * akt. Sound-Byte nach RØ
        BLWP @VSBW          * in Soundliste eintragen
        INC  RØ             * VDP-Adresse + 1
        MOV  R2,R1          * Frequenzwert holen
        SLA  R1,4           * die 8 hoeherwertigen Bits isolieren
        BLWP @VSBW          * und ins VDP-RAM schreiben
        INCT @SBYTE         * akt. Sound-Byte um zwei erhoehen
        INCT @SCOUNT        * Byte-Zaehler um zwei erhoehen
        B    @COMP          * Ueberpruefen ob Buffer voll und BASIC-Rueckkehr

* Routine generiert Frequenzeintrag fuer Soundgenerator 2 (siehe FREQ1)
FREQ2
        MOV  R11,R1Ø
        BL   @NUMBER
        BL   @DIVIDE
        MOV  R1,R2
        SLA  R1,12
        SRL  R1,4
        AI   R1,>AØØØ
        MOV  @SBYTE,RØ
        BLWP @VSBW
        INC  RØ
        MOV  R2,R1
        SLA  R1,4
        BLWP @VSBW
        INCT @SBYTE
        INCT @SCOUNT
        B    @COMP

* Routine generiert Frequenzeintrag fuer Soundgenerator 3 (siehe FREQ1)
FREQ3
        MOV  R11,R1Ø
        BL   @NUMBER
        BL   @DIVIDE
        MOV  R1,R2
        SLA  R1,12
        SRL  R1,4
        AI   R1,>CØØØ
        MOV  @SBYTE,RØ
        BLWP @VSBW
        INC  RØ
        MOV  R2,R1
        SLA  R1,4
        BLWP @VSBW
        INCT @SBYTE
        INCT @SCOUNT
        B    @COMP

* Routine generiert Frequenzeintrag fuer den Geraeuschgenerator
FREQ4
        MOV  R11,R1Ø        * Ruecksprung sichern
        BL   @NUMBER        * Zahl in RØ holen
        NEG  RØ             * negieren
        MOV  RØ,RØ          * auf null pruefen
        JEQ  FREQ41         * Wenn null, dann Sprung nach FREQ41
        CI   RØ,B           * mit B vergleichen
        JLE  FREQ42         * falls kleiner oder gleich, Sprung nach FREQ42
FREQ41  LI   RØ,>1EØØ       * Fehlermeldung BAD VALUE
        BLWP @ERR           * ausgeben und zurueck ins BASIC
FREQ42  DEC  RØ             * - 1
        SLA  RØ,8           * Low-Byte RØ nach High-Byte RØ schieben
        AI   RØ,>EØØØ       * OP-Code dazu
        MOV  RØ,R1          * nach R1 (zum schreiben)
        MOV  @SBYTE,RØ      * akt. Sound-Byte nach RØ
        BLWP @VSBW          * und in die Soundliste eintragen
        INC  @SBYTE         * akt. Sound-Byte + 1
```

```
       INC  @SCOUNT       * Byte-Zaehler + 1
       B    @COMP         * Ueberpruefen ob Buffer voll und BASIC-Rueckkehr
* Routine generiert die Lautstaerke fuer Soundgenerator 1
LAUT1
       MOV  R11,R1Ø       * Ruecksprung retten
       BL   @NUMBER       * Zahl nach RØ laden
       CI   RØ,31         * RØ kleiner 31?
       JL   LAUT11        * Ja, dann Sprung nach LAUT11
       LI   RØ,>1EØØ      * sonst Fehlermeldung BAD VALUE
       BLWP @ERR          * und Ruecksprung ins BASIC
LAUT11 SLA  RØ,7          * Low-Byte nach High-Byte schieben (das letzte
*                         * Bit wird uebersehen)
       AI   RØ,>9ØØØ      * OP-Code dazu
       MOV  RØ,R1         * nach R1 (zum schreiben ins VDP-RAM)
       MOV  @SBYTE,RØ     * VDP-Adresse in RØ
       BLWP @VSBW         * schreiben
       INC  @SBYTE        * akt. Sound-Byte incrementieren
       INC  @SCOUNT       * Byte-Zaehler incrementieren
       B    @COMP         * Ueberpruefen ob Buffer voll und BASIC-Rueckkehr

* Routine generiert die Lautstaerke fuer Soundgenerator 2 (siehe LAUT1)
LAUT2
       MOV  R11,R1Ø
       BL   @NUMBER
       CI   RØ,31
       JL   LAUT21
       LI   RØ,>1EØØ
       BLWP @ERR
LAUT21 SLA  RØ,7
       AI   RØ,>BØØØ
       MOV  RØ,R1
       MOV  @SBYTE,RØ
       BLWP @VSBW
       INC  @SBYTE
       INC  @SCOUNT
       B    @COMP

* Routine generiert die Lautstaerke fuer Soundgenerator 3 (siehe LAUT1)
LAUT3
       MOV  R11,R1Ø
       BL   @NUMBER
       CI   RØ,31
       JL   LAUT31
       LI   RØ,>1EØØ
       BLWP @ERR
LAUT31 SLA  RØ,7
       AI   RØ,>DØØØ
       MOV  RØ,R1
       MOV  @SBYTE,RØ
       BLWP @VSBW
       INC  @SBYTE
       INC  @SCOUNT
       B    @COMP

* Routine generiert die Lautstaerke fuer den Geraeuschgenerator (siehe LAUT1)
LAUT4
       MOV  R11,R1Ø
       BL   @NUMBER
       CI   RØ,31
       JL   LAUT41
       LI   RØ,>1EØØ
       BLWP @ERR
LAUT41 SLA  RØ,7
       AI   RØ,>FØØØ
       MOV  RØ,R1
       MOV  @SBYTE,RØ
       BLWP @VSBW
       INC  @SBYTE
       INC  @SCOUNT
       B    @COMP

* Routine generiert die Dauer des vorher definierten Eintrags und initialisiert
* einen neuen Soundlisteneintrag
DAUER
       MOV  R11,R1Ø       * Ruecksprung retten
       BL   @NUMBER       * Zahl in RØ ablegen
       MOV  RØ,R1         * Zahl sichern
       SLA  R1,B          * Low-Byte nach High-Byte schieben
       MOV  @SBYTE,RØ     * akt. Sound-Byte nach RØ (fuer VDP-Zugriff)
       INC  @SBYTE        * Sound-Byte incrementieren
       BLWP @VSBW         * schreiben
       MOV  @LAENGE,RØ    * Adresse des ersten Bytes des Eintrags (Laengen-Byte)
       MOV  @SCOUNT,R1    * Byte-Zaehler nach R1
       SLA  R1,B          * und B Bit nach links schieben
       BLWP @VSBW         * ins VDP-RAM schreiben
       MOV  @SBYTE,@LAENGE * neues Laengenbyte erhaelt Wert des Sound-Bytes
       INC  @SBYTE        * und erhoehen
       CLR  @SCOUNT       * Byte-Zaehler loeschen
       B    @COMP         * Pruefen ob Buffer voll ist und BASIC-Ruecksprung

* Routine generiert einen Sprung-Eintrag zu einem vorher definierten Label
SPRUNG
       MOV  R11,R1Ø       * Ruecksprung retten
       BL   @STRING       * String in String-Buffer einlesen
       BL   @MARKE        * Mit schon definierten Labels vergleichen
       MOV  RØ,R2         * Sprungadresse in RØ nach R2 kopieren
       MOV  @SBYTE,RØ     * akt. Sound-Byte in RØ (fuer VDP-Zugriff)
       DEC  RØ            * kein Laengenbyte
       CLR  R1            * null = OP-Code fuer Sprung
       BLWP @VSBW         * schreiben
       INC  RØ            * VDP-Adresse erhoehen
       MOV  R2,R1         * Sprungadresse
       BLWP @VSBW         * hoeherwertiges Byte schreiben
       INC  RØ            * VDP-Schreibadresse erhoehen
       SWPB R1            * Bytes vertauschen
       BLWP @VSBW         * und niederwertiges Byte schreiben
       INC  @SBYTE        * akt. Sound-Byte +3
       INCT @SBYTE        *
       MOV  @SBYTE,@LAENGE * Adr. des Laengenbytes erhaelt Wert des Sound-Bytes
       INC  @SBYTE        * akt. Soundbyte incrementieren
       CLR  @SCOUNT       * Byte-Zaehler loeschen
       B    @COMP         * Ueberpruefen ob Buffer voll ist und Ruecksprung

* interruptgesteuerte Soundlistenverarbeitung initialisieren
XSOUND
       MOV  R11,R1Ø       * Ruecksprung retten
       BL   @STRING       * String in String-Buffer holen
       BL   @MARKE        * String mit Labels vergleichen, Adresse in RØ
```

```
MOV  RØ,@>83CC     * Adresse der Liste nach >83CC
MOVB @HØ1,@>83CE   * Soundlistenverarbeitung ausloesen
SOCB @HØ1,@>83FD   * VDP-RAM Flag setzen (1=VDP-RAM / Ø=GROM)

B    *R1Ø          * Ruecksprung ins BASIC

* Stoppt die interruptgesteuerte Soundlistenverarbeitung
STOP
     CLR  RØ            * Soundlistenverarbeitung stoppen
     MOVB RØ,@>83CE     *
     LI   RØ,>9FØØ      * Lautstaerke Generator 1 auf 3Ø
     MOVB RØ,@SOUND     *
     LI   RØ,>BFØØ      * Lautstaerke Generator 2 auf 3Ø
     MOVB RØ,@SOUND     *
     LI   RØ,>DFØØ      * Lautstaerke Generator 3 auf 3Ø
     MOVB RØ,@SOUND     *
     LI   RØ,>FFØØ      * Lautstaerke des Geraeuschgenerators auf 3Ø
     MOVB RØ,@SOUND     *
     RT                 * Ruecksprung

     END
```

```
1 REM********************
2 REM*                  *
3 REM*    SOUND-DEMO    *
4 REM*   ------------   *
5 REM*                  *
6 REM* 6.7.85 T. MIELKE *
7 REM*                  *
8 REM********************
9 REM
1Ø CALL LINK("SLSTOP")
11 CALL LINK("LOESCH")
12 CALL LINK("LABEL","RAKETE")
13 CALL LINK("FREQ4",-8)
14 CALL LINK("LAUT4",Ø)
15 FOR I=4ØØ TO 18ØØ STEP 1Ø
16 CALL LINK("FREQ3",I)
17 CALL LINK("LAUT4",(I-4ØØ)/46)
18 CALL LINK("DAUER",1)
19 NEXT I
2Ø CALL LINK("LAUT4",3Ø)
21 CALL LINK("DAUER",Ø)
22 CALL LINK("XSOUND","RAKETE")
23 PRINT "BITTE TASTE DRUECKEN!": : : : :
24 CALL KEY(Ø,A,B)
25 IF B<>1 THEN 24
26 CALL LINK("XSOUND","RAKETE")
27 GOTO 24
```


DELETE

Mit diesem Programm ist es möglich, innerhalb von nicht 'mal einer Sekunde beliebige Programmblökke aus einem XBasic-Programm zu löschen!
Nach dem Laden durch >CALL INIT >CALL LOAD ("DSK1.$X DELETE") erfolgt der Aufruf mit CALL LINK ("DELETE",START-LINE,ENDLINE) aus dem XBasic-Direktmodus!
STARTLINE und END-LINE können beliebige gültige Zeilennummern sein, sie brauchen im Programm nicht zu existieren.
Im Gegensatz zu anderen Programmen bleiben alle LIST und EDIT Funktionen voll erhalten, und das Programm ist nach dem Aufruf voll funktionsfähig, d.h. es kann gestartet, gelistet, gespeichert und editiert werden.
Allerdings wird keine Speicherplatzoptimierung durchgeführt. Wird diese gewünscht, so ist das Programm als MERGE-Datei abzuspeichern, der Speicher zu löschen und das Programm via MERGE wieder einzuladen.
Notwendige Gerätekonfiguration: TI 99/4A-Konsole, XBasic, 32 K RAM, mind. 1 Laufwerk. Zur Eingabe des Quellcodes zusätzlich: Editor/Assembler.

Peter Ulbrich

```
************************************************************************
*                                                                      *
*    DELETE -- Loescht Programmsegmente                                *
*                                                                      *
*    TMS-9900 ASSEMBLER, Version XBASIC                                *
*                                                                      *
*    Programmname: @XDELETE / @XDELETES                                *
*                                                                      *
*    1985-06-18 by Peter Ulbrich                                       *
*                                                                      *
*    Aufruf aus XBASIC-Direktmodus mit:                                *
*    CALL LINK("DELETE",STARTLINE,ENDLINE)                             *
*                                                                      *
*    Das Assembler-Programm loescht den angegebenen Block aus dem      *
*    XBASIC-Programm. Ist STARTLINE groesser als ENDLINE, erfolgt eine *
*    Fehlermeldung. Das XBASIC-Programm ist nach dem Aufruf voll funk- *
*    tionsfaehig, nur die geloeschten Zeilen stehen eben nicht mehr zur *
*    Verfuegung. Allerdings wird keine Speicheplatzoptimierung vorge-  *
*    nommen. Wird diese gewuenscht, so ist folgendermassen vorzugehen: *
*    > SAVE DSK1.Name,MERGE      )  Dabei bedeutet                     *
*    > NEW                       )  Name                               *
*    > MERGE DSK1.Name           )  irgendein gueltiger Dateiname      *
*                                                                      *
************************************************************************


       DEF DELETE

* XBASIC-Equates

NUMREF EQU  >200C          Uebernimmt numerische Parameter
XMLLNK EQU  >201B          ROM-Routine:
CFI    EQU  >12BB          Floating Point --> Integer
ERR    EQU  >2034          Fehlerbehandlung:
ERRBA  EQU  >1C00          Bad Argument
ERRIAL EQU  >1F00          Incorrect Argument List

* Sonstige Equates

ARGNMR EQU  >B312          Argumentenanzahl
FAC    EQU  >834A          Floating Point Accumulator
LINBEG EQU  >B330          Beginn der Zeilennummerntabelle
LINEND EQU  >B332          Ende der Zeilennummerntabelle
NEXT   EQU  >0070          Next
GPLWS  EQU  >B3E0          GPL-Workspace
STATUS EQU  >837C          GPL Status Byte

* Workspace, Konstante

WS     BSS  >20            Eigene Arbeitsregister
TWO    DATA 2

       EVEN

* Unterprogramme
* --------------

* Error - Routine

ERR1   LI   R0,ERRBA      Fehlerbehandlung
       JMP  ERROR
ERR2   LI   R0,ERRIAL     Fehlerbehandlung
ERROR  BLWP @ERR          Fehlermeldung ausgeben und Programm stoppen

* Parameter Uebernahme

GETPAR MOV  R11,R10            Ruecksprung sichern
       CB   @ARGNMR,@TWO+1     Argumentenanzahlkontrolle
       JNE  ERR2          Anzahl<>2 --> Error
       CLR  R0
       LI   R1,1          Zeiger auf 1. Parameter
       BLWP @NUMREF       1. Parameter holen (STARTLINE)
       BLWP @XMLLNK
       DATA CFI           und in Integer verwandeln
       MOV  @FAC,RB       1. Parameter ist nun in RB
       CLR  R0
       LI   R1,2          Zeiger auf 2.Parameter
       BLWP @NUMREF       2. Parameter holen  (ENDLINE)
       BLWP @XMLLNK
       DATA CFI           und in Integer verwandeln
       MOV  @FAC,R9       2. Parameter ist nun in R9
       C    RB,R9         Vergleich
       JGT  ERR1          1. Parameter ist groesser --> Fehler
       B    *R10          Zurueck zum Aufrufer

* Basic-Return

BSCRTN CLR  R0
       MOVB R0,@STATUS    Statusbyte loeschen
       LWPI GPLWS         GPL Arbeitsregister laden
       B    @NEXT         Zurueck nach Basic

* Hauptprogramm
* =============

DELETE LWPI WS            Eigene Arbeitsregister laden
       BL   @GETPAR       Parameter holen
       CLR  R10           Counter fuer Anzahl der zu loeschenden Bytes
       CLR  R5            Flag, ob das gesamte Programm zu loeschen ist
       MOV  @LINEND,R0    Ende der Zeilennummerntabelle holen
       DEC  R0
LOOP1  DECT R0            Aktueller Pointer auf Zeilennummer
       C    R0,@LINBEG    Vergeleich mit Beginn
       JLT  RETURN        Kleiner, also nichts zu loeschen --> XBASIC
       CLR  R1            )
       MOVB *R0,R1        ) nun die Zeilennummer
       INC  R0            ) byteweise
       CLR  R2            ) holen
       MOVB *R0,R2        )
       DEC  R0            )
       SRL  R2,B          )
       A    R2,R1         ) liegt jetzt in R1
       C    RB,R1         Vergleich mit STARTLINE
       JEQ  CONT1         Gleich --> Es ist etwas zu loeschen --> CONT1
       JLT  L1            Kleiner --> Erst pruefen: Nach L1
       DECT R0            Pointer erniedrigen
```

```
        INC  R5          Flag setzen! ( Programm wird nicht ganz geloescht)
        JMP  LOOP1       und Schleife durchlaufen
L1      C    R9,R1       Vergleich von ENDLINE mit der aktuellen
                         Zeilennummuer
        JLT  RETURN      Kleiner, also gibt es nichts zu loeschen -->
                         XBASIC
CONT1   NOP              Weiter
        DECT R0          Pointer erniedrigen
        AI   R10,4       Bytezaehler erhoehen
LOOP2   DECT R0          Aktueller Pointer auf Zeilennummer
        C    R0,@LINBEG  Mit Beginn vergleichen
        JLT  CONT2       Kleiner --> CONT2 mit Pruefung
        CLR  R1          )
        MOVB *R0,R1      )
        INC  R0          )  Zeilennummer byteweise holen
        CLR  R2          )
        MOVB *R0,R2      )
        DEC  R0          )
        SRL  R2,8        )
        A    R2,R1       ) liegt jetzt in R1
        C    R9,R1       Vergleich ENDLINE mit aktueller Zeilennummer
        JLT  CONT3       Kleiner --> CONT3
        DECT R0          Pointer erniedrigen
        AI   R10,4       Bytezaehler erhoehen
        JMP  LOOP2       und LOOP2 durchlaufen
CONT2   MOV  R5,R5       Pruefen, ob Flag gesetzt
        JEQ  CONT5       Nein, gesamtes Programm loeschen --> CONT5
CONT3   MOV  R0,R7       Hier wird nur ein Teil geloescht, also R0 kopieren
        A    R10,R7      R7 um R10 erhoehen
LOOP3   LI   R6,4        Schleifencounter laden
L2      MOVB *R0+,*R7+   ) 4 Bytes von der aktuellen Adresse
        DEC  R6          ) um den Wert von R10
        MOV  R6,R6       ) verschieben
        JNE  L2          )
        AI   R0,-8       Adressen erniedrigen
        AI   R7,-8
        C    R0,@LINBEG  Schon am Beginn?
        JLT  CONT4       Ja, weiter bei CONT4
        JMP  LOOP3       Nein, LOOP3 durchlaufen
CONT4   MOV  @LINBEG,R5  Beginn holen
        A    R10,R5      um den Wert von R10 hochsetzten
        MOV  R5,@LINBEG  und wieder speichern
        JMP  RETURN      Fertig, jetzt zurueck!
CONT5   MOV  @LINEND,R5  Gesamtes Programm loeschen, dazu Ende holen
        MOV  R5,@LINBEG  und auf Beginn legen
RETURN  BL   @BSCRTN     Basicreturn

        END
```



BILD-SCHIRM SPEICHERN

Um das Programm in TI-Basic laufen zu lassen, genügt es, nicht die Equates zu ändern, da das VDP-RAM in TI-Basic anders genutzt wird als in XBas. Call Link ("COPY",X) speichert den Bildschirm einschließlich Farben (außer Bildschirmfarbe), Sprites, Motions und Magnify. "LCOPY" holt ein abgespeichertes Bild vom Speicher und bringt es auf den Bildschirm. Der Parameter X gibt spezifiziert den Speicher. X=1:RAMspeicherbereich 1, X=2:RAMspeicherbereich 2, X=Device-Name. CALL LINK("COPY", "DSK1.BILD1") speichert also den Bildschirm als File mit Namen "BILD1" auf Diskette (10 Sektoren) ab. Leider habe ich keine Möglichkeit gefunden, die Bildschirmfarbe irgendwo auszulesen. Da beim Einlesen auch der Bereich >0370–>03EF im VDP-RAM geändert wird, ist es möglich, daß nach Drücken von "REDO" ein anderer Inhalt vorhanden ist. Die DSRLNK-Routine ist fast identisch mit der des Ed/Ass. Ich habe sie „von Hand" disassembliert und leicht geändert. Der Device-Name wird in den Bereich ab >24EA kopiert. Dort sind noch ein paar Bytes frei, da die Utilities bei >24E9 aufhören und der Loader-Pointer (>2002) ohne Maschinenprogramm schon auf >24FA steht.

Heinrich Acker

```
*****************************************
*                                       *
*  CALL LINK (COPY,X)  => ABSPEICHERN   *
*                                       *
*  CALL LINK (LCOPY,X) => SCHREIBEN     *
*                                       *
*    X=1 :BILD 1        X=2 :BILD 2     *
*                                       *
*    X=STRING :DEVICE NAME              *
*                                       *
*   4750 BYTES                          *
*                                       *
*   !!!  NUR FUER EXT.8ASIC  !!!        *
*                                       *
*  12.Ø8.1985 BY HEINRICH ACKER     V1 *
*****************************************
       IDT  'XCOPY1Ø'
       DEF  COPY,LCOPY
VMBW   EQU  >2Ø24          *
VSBW   EQU  >2Ø2Ø          *
VMBR   EQU  >2Ø2C          *
VSBR   EQU  >2Ø28          *
VWTR   EQU  >2Ø3Ø          *
GPLWS  EQU  >83EØ          *
STATUS EQU  >837C          *
FAC    EQU  >834A          *
NUMREF EQU  >2ØØC          *
STRREF EQU  >2Ø14          *
XMLLNK EQU  >2Ø18          *
CFI    EQU  >12B8          *
PAD    EQU  >83ØØ          * ARGUMENT BEZEICHNER LISTE
ERR    EQU  >2Ø34          *
ERRSYN EQU  >Ø3ØØ          * SYNTAX ERROR
ERR8V  EQU  >1EØØ          * BAD VALUE
ERRIAL EQU  >1FØØ          * INCORRECT ARGUMENT LIST
ERRIO  EQU  >24ØØ          * I/O ERROR
BUFSCR 8SS  1952           * 8UFFER FUER 8ILD 1
       DATA >EØ1C          * UND VDPR 1 >83D4 (MAGNIFY) UND >837A (MOTION)
       8SS  1952           * BUFFER FUER BILD 2
       DATA >EØ1C          * UND VDPR 1 >83D4 (MAGNIFY) UND >837A (MOTION)
PA8    EQU  >Ø9ØØ          * ANFANG DES PABS IM VDP
PDATA  DATA >ØØ18,>ØØØØ,>DCDC,>ØØØØ     * VARIA8LE 22Ø,INTERNAL
       8YTE >ØØ            *
8UFSTR 8SS  >29            * PARAMETER STR-8UFFER (41 8YTES,1.BYTE=LAENGE)
STRLAN 8YTE >28            * MAX. STRING-LAENGE (4Ø BYTES)
       EVEN                *
MYWS   8SS  >2Ø            *
**************************************************************************
*      S P E I C H E R N      *
******************************
COPY   LWPI MYWS
       BL   @GETVAL
       CI   RØ,>ØØØØ       * PRUEFE OB BILD IN MEMORY ODER DEVICE
       JEQ  COPYD
COPYM  CLR  RØ
       LI   R2,88Ø
       8LWP @VM8R
       LI   RØ,1ØØ8
       AI   R1,88Ø
       LI   R2,1Ø72
       BLWP @VMBR
       AI   R1,1Ø72
       MOV8 @>83D4,*R1+    * MAGNIFY ABSPEICHERN
       MOV8 @>837A,*R1     * ANZAHL DER BEWEGTEN SPRITES A8SPEICHERN
       B    @ENDE
COPYD  LI   RØ,>Ø82Ø
       MOV8 @>83D4,R1
       BLWP @VSBW          * MAGNIFY IN VDP >Ø82Ø ABSPEICHERN
       INC  RØ
       MOV8 @>837A,R1
       BLWP @VSBW          * ANZAHL DER 8EW.SPRITES IN VDP >Ø821 ABSPEICHERN
       LI   R4,>Ø3ØØ       * R4 (HB) : WRITE OP-CODE
       LI   R1,>1AØØ       * OUTPUT-CODE
       BL   @CPA8
       B    @ENDE
**************************************************************************
*       L A D E N            *
******************************
LCOPY  LWPI MYWS
       BL   @GETVAL
       CI   RØ,>ØØØØ       * PRUEFE OB BILD IN MEMORY ODER DEVICE
       JEQ  LCOPYD
LCOPYM CLR  RØ
       LI   R2,88Ø
       BLWP @VMBW
       LI   RØ,1ØØ8
       AI   R1,88Ø
       LI   R2,1Ø72
       BLWP @VM8W
       AI   R1,1Ø72
       MOVB *R1,@>83D4     * MAGNIFY IN >83D4
       LI   RØ,1
       MOV8 *R1+,RØ
       SWP8 RØ
       8LWP @VWTR          * UND VDP-REG 1 SCHREI8EN
       MOV8 *R1,@>837A     * ANZAHL DER 8EWEGTEN SPRITES SCHREI8EN
       8    @ENDE
LCOPYD LI   R4,>Ø2ØØ       * R4 (HB) : READ OP-CODE
       LI   R1,>1CØØ       * INPUT-CODE
       BL   @CPA8
       LI   RØ,>Ø82Ø
       BLWP @VSBR
       MOV8 R1,@>83D4      * MAGNIFY IN >83D4
       LI   RØ,1
       MOVB R1,RØ
       SWPB RØ
       BLWP @VWTR          * UND VDP-REG 1 SCHREI8EN
       LI   RØ,>Ø821
       8LWP @VS8R
       MOV8 R1,@>837A      * ANZAHL DER 8EWEGTEN SPRITES SCHREI8EN
       8    @ENDE
**************************************************************************
*       S U 8 S              *
******************************
GETVAL CLR  RØ
       CLR  R1
       MOV8 @>8312,R1      * >8312 ENTHAELT ANZAHL DER PARAMETER
       SWP8 R1
```

```
        CI   R1,>0001       * PRUEFE OB 1 PARAMETER
        JEQ  CHBEZ          * FALLS JA --> CHeck BEZeichner
        LI   R0,ERRSYN
        BLWP @ERR           * SONST 'SYNTAX ERROR'
CHBEZ   CLR  R2
        MOVB @PAD,R2        * >8300 ENTHAELT BEZEICHNER DES 1.PARAMETERS
        CI   R2,>0000       * PRUEFE OB NUM.AUSDRUCK
        JEQ  NUREF
        CI   R2,>0200       * PRUEFE OB NUM.VARIABLE
        JEQ  NUREF
        CI   R2,>0100       * PRUEFE OB STR.AUSDRUCK
        JEQ  STREF
        CI   R2,>0300       * PRUEFE OB STR.VARIABLE
        JEQ  STREF
        LI   R0,ERRIAL
        BLWP @ERR           * SONST 'INCORRECT ARGUMENT LIST'
NUREF   BLWP @NUMREF        * BRINGT NUM.PARAMETER INS FAC
        BLWP @XMLLNK        * WANDELT FAC IN GANZE ZAHL UM
        DATA CFI
        LI   R1,BUFSCR      * BUFFER FUER BILD 1 IN R1
        MOV  @FAC,R0        * PARAWERT IN R0
        CI   R0,>0001       * PRUEFE OB PARAWERT=1
        JEQ  RETUR
        AI   R1,1954        * BUFFER FUER BILD 2 IN R1
        CI   R0,>0002       * PRUEFE OB PARAWERT=2
        JEQ  RETUR
        LI   R0,ERRBV
        BLWP @ERR           * SONST 'BAD VALUE'
STREF   MOVB @STRLAN,@BUFSTR  * MAX. STR-LAENGE IN BUFSTR
        LI   R2,BUFSTR
        BLWP @STRREF        * STRING-PARAMETER IN BUFSTR (1.BYTE=STR-LAENGE)
RETUR   RT
CPAB    MOVB R1,@PDATA+1    * INPUT/OUTPUT-CODE IN PAB-DATA
        LI   R0,PAB
        LI   R1,PDATA
        LI   R2,50
        BLWP @VMBW          * PDATA INS PAB IM VDP
        LI   R3,PAB+9
        MOV  R3,@>8356      * >8356 MUSS VDP-ADR, WO NAMENSL.STEHT  ENTHALTEN
        BLWP @DSRLNK        * DATEI OEFFNEN
        DATA 8
        JEQ  ERIO           * FALLS EQ-BIT=1 --> I/OERR
        MOVB R4,R1          * R1 (HB) : READ BZW WRITE OP-CODE
        BLWP @VSBW          * INS PAB
        CLR  R1             * R1 : AKTUELLE VDP-ADR.,AB DER KOPIERT WIRD
        CLR  R4             * R4 : SCHLEIFENZAEHLER
CPAB1   MOV  R3,@>8356
        BLWP @DSRLNK        * DATEN LESEN/SCHREIBEN
        DATA 8
        JEQ  ERIO
        INC  R4             * SCHLEIFENZAEHLER ERHOEHEN
        CI   R4,4
        JEQ  CPAB3
        CI   R4,B
        JEQ  CPAB4
        CI   R4,9
        JEQ  CPAB5
        AI   R1,220         * VDP-ADR. UM 220 ERHOEHEN
CPAB2   LI   R0,PAB+2
        BLWP @VSBW          * NEUE VDP-ADR. INS PAB
        INC  R0
        SWPB R1
        BLWP @VSBW
        SWPB R1
        JMP  CPAB1
CPAB3   LI   R1,100B
        JMP  CPAB2
CPAB4   LI   R0,PAB+5
        LI   R1,>C200       * R1 (HB) : 194
        BLWP @VSBW
        LI   R1,188B
        JMP  CPAB2
CPAB5   LI   R0,PAB
        LI   R1,>0100       * R1 (HB) : CLOSE OP-CODE
        BLWP @VSBW          * INS PAB
        MOV  R3,@>8356
        BLWP @DSRLNK        * DATEI SCHLIESSEN
        DATA 8
        JNE  RETUR          * FALLS EQ-BIT=0 (KEIN FEHLER) --> RETUR
ERIO    LI   R0,ERRIO
        BLWP @ERR           * SONST 'I/O ERROR'
ENDE    CLR  R0
        MOVB R0,@STATUS
        LWPI GPLWS
        B    @>0070         * RUECKSPRUNG INS XBASIC
*************************************************************************
*       DSR-ROUTINE FUER XBAS.     *                            *       *
*       ----------------------     *    R13 : ALT WP            *       *
*                                  *    R14 : ALT PC            *       *
* BEI ERROR :                      *    R15 : STATUS            *       *
* EQ-BIT IM STATUS-REG. GESETZT    *                            *       *
* R0 (HB) : FEHLERCODE (0-7)       ******************************       *
* >8304 : VDP-ADR,WO PAB ANFANG    *                            *       *
*       MINUS >4  (OFFSET)         *    >000D : >AA    (HEADER) *       *
*                                  *    >03B4 : >2000           *       *
* KEIN ERROR :                     *    >0CF7 : >2E    (ASC".") *       *
* EQ-BIT NICHT GESETZT             *                            *       *
*                                  *                            *       *
*************************************************************************
DSRWS   BSS  >20
DSRLNK  DATA DSRWS,$+2    * WP : DSRWS , PC : MOMENTANER PC + 2
        MOV  *R14+,R5     * SICHERE 8 IN R5 (DATA 8  AUS DSRLNK-AUFRUF)
        SZCB @>03B4,R15   * LOESCHT EQ-BIT IN 'STATUS'-REG AUF JEDEN FALL
        MOV  @>8356,R0    * R0 : VDP-ADR,WO FILE-DESCRIPTOR LAENGE STEHT
        MOV  R0,R9
        AI   R9,>FFF8     * R9 : VDP-ADR,WO I/O FLAG STEHT
        BLWP @>2028       * (VSBR)
        MOVB R1,R3        * R1,R3 (HB) : FILE-DESCRIPTOR LAENGE
        SRL  R3,8         * R3 : FILE-DESCRIPTOR LAENGE
        SETO R4           * R4 : >FFFF
        LI   R2,>24EA     * STARTADR. FUER FILE-DESCRIPTOR KOPIE
DSR1    INC  R0           * DSR1 KOPIERT AUS VDP DEVICE-NAME IN >24EA+
        INC  R4           *
        CI   R4,>0007     * PRUEFE OB DEVICE-NAME >7
        JGT  IOERR2       * FALLS JA --> IOERR2
        C    R4,R3        * PRUEFT OB FILE-DESCR. VOLLSTAENDIG
        JEQ  DSR2         * JA --> DSR2
        BLWP @>2028       * ANDERNFALLS WEITERLESEN (VSBR)
```

```
        MOVB R1,*R2+        * KOPIERBEFEHL
        CB   R1,@>ØCF7      * PRUEFE OB BYTE="."
        JNE  DSR1           * NEIN --> DSR1
DSR2    MOV  R4,R4          * R4 : DEVICE-NAME LAENGE = LAENGE OHNE EVENT."."
        JEQ  IOERR2         * FALLS R4=Ø --> IOERR2
        CLR  @>83DØ
        MOV  R4,@>8354      * >8354 : DEVICE-NAME LAENGE
        INC  R4             * R4 : DEVICE-NAME LAENGE + 1
        A    R4,@>B356      * >B356 : ZEIGT AUF VDP-ADR. HINTER LETZTES BYTE
*                           * VON DEVICE-NAME (EVENT.WO "." STEHT)
DSR3    LWPI >B3EØ          **** G P L - W O R K S P A C E ****
        CLR  R1
        LI   R12,>ØFØØ      * R12 : >ØFØØ   CRU-BASIS ADR.
DSR4    MOV  R12,R12
        JEQ  DSR5           * FALLS R12=Ø --> DSR5
        SBZ  Ø              * SCHALTE CRU (WIEDER) AUS
DSR5    AI   R12,>Ø1ØØ      * CRU-BASIS UM >Ø1ØØ ERHOEHEN
        CLR  @>83DØ
        CI   R12,>2ØØØ      * PRUEFE OB GESAMTE CRU-ADR. SCHON ABGEKLAPPERT
        JEQ  IOERR1         * FALLS JA --> IOERR1
        MOV  R12,@>B3DØ     * >B3DØ : CRU-BASIS ADR.
        SBO  Ø              * SONST SCHALTE CRU EIN
        LI   R2,>4ØØØ
        CB   *R2,@>ØØØD     * PRUEFE OB AN >4ØØØ >AA STEHT (HEADER)
        JNE  DSR4           * FALLS NEIN --> DSR4
        A    @DSRWS+1Ø,R2 * ALT R5+R2=    R2 : >4ØØB  BEI (DATA B)
        JMP  DSR7           * >4ØØ8 : POINTER AUF NAMENSTABELLE IM PERI-SYS.
DSR6    MOV  @>B3D2,R2      * R2 : WEITERE POINTER
        SBO  Ø              * CRU EINSCHALTEN
DSR7    MOV  *R2,R2         * R2 : (WEITERE) POINTER (BEIM 1.MAL VON >4ØØB)
        JEQ  DSR4           * FALLS R2=Ø --> DSR4
        MOV  R2,@>83D2      * >83D2 : (WEITERE) POINTER DES PERI-SYS
        INCT R2             * R2 : POINTER + 2
        MOV  *R2+,R9        * R9 : @(POINTER+2) = STARTADR. DES PERI-SYS
        MOVB @>8355,R5      * R5 (HB): DEVICE-NAME LAENGE
        JEQ  DSR9           * FALLS R5=Ø --> DSR9
        CB   R5,*R2+        * PRUEFE OB DEVICE-NAME LAENGE STIMMT
        JNE  DSR6           * FALLS NEIN --> DSR6
        SRL  R5,B           * R5 (LB): DEVICE-NAME LAENGE
        LI   R6,>24EA       * R6 : >24EA
DSR8    CB   *R6+,*R2+      * PRUEFE OB DEVICE-NAME STIMMT (BYTEWEISE VERGL.)
        JNE  DSR6           * FALLS IRGENDEIN BYTE NICHT STIMMT --> DSR6
        DEC  R5             * R5 : NOCH ZU PRUEFENDE BYTES
        JNE  DSR8           * FALLS NOCH NICHT ALLE GEPRUEFT --> DSRB
DSR9    INC  R1
        BL   *R9            * SPRINGE INS PERIPHERIE-BETRIEBSSYSTEM
        JMP  DSR6           * PERI-SYS LAESST BEFEHL UEBERSPRINGEN
        SBZ  Ø              * CRU AUSSCHALTEN
        LWPI DSRWS          **** D S R L N K - W O R K S P A C E ****
        MOV  R9,RØ          * RØ : VDP-ADR.,WO I/O FLAG STEHT
        BLWP @>2Ø28         * I/O FLAG LESEN  (VSBR)
        SRL  R1,13          * R1 : I/O FEHLER-CODE
        JNE  IOERR3         * FALLS R1<>Ø --> IOERR3
        RTWP
IOERR1  LWPI DSRWS
IOERR2  CLR  R1
IOERR3  SWPB R1             * R1 (HB) : FEHLERCODE
        MOVB R1,*R13        * *R13 IST RØ VON HAUPTPROGRAMM-WS
        SOCB @>Ø3B4,R15     * SETZT EQ-BIT IN 'STATUS'-REG AUF JEDEN FALL
        AI   R9,>FFFB       * R9 : PAB-ANFANG MINUS >4 IM VDP
        MOV  R9,@>83Ø4      *
        RTWP
*****************************************************************************
        END
```

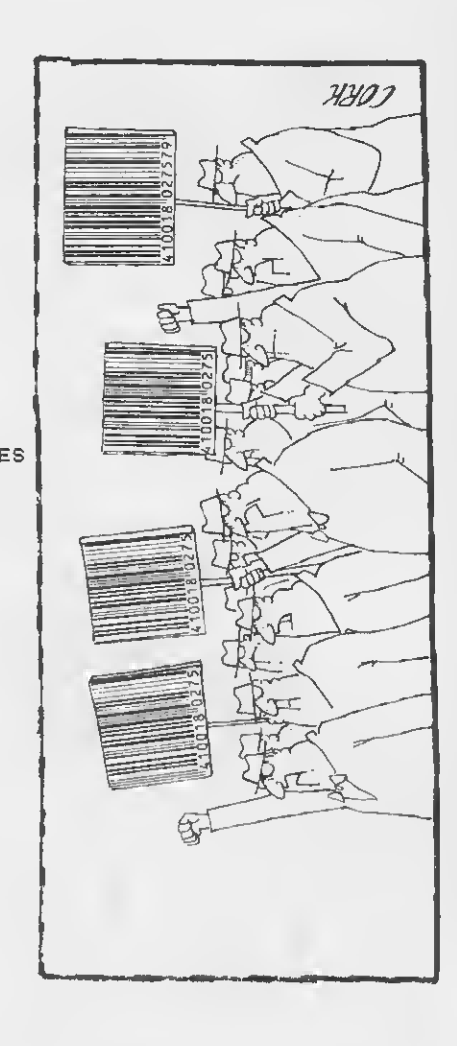
SCROLL IN AUSSCHNITTEN

Das Programm ist für Extended-Basic geeignet. Um das Programm in TI-Basic laufen zu lassen, brauchen nur die Equates geändert werden. Call Link ("SCROLL", A,B,C[,D,E]) scrollt einen bestimmten Teil des Bildschirms nach unten oder nach oben. Dies ist unter anderem zur Darstellung von Tabellen nützlich; die Kopfzeilen einer Tabelle bleiben so immer erhalten. Die Parameter A,B, C,D,E definieren einen Ausschnitt des Bildschirms. A ist die oberste, B die unterste Zeile, D ist die erste, E die letzte Spalte. D und E sind optional. Falls sie nicht eingegeben werden ist D=1 und E=32. C gibt an, um wieviel Zeilen gescrollt wird. Bei positivem C wird nach unten, bei negativem C nach oben gescrollt.

Heinrich Acker

```
*************************************
*                                   *
*  CALL LINK("SCROLL",A,B,C [,D,E]) *
*                                   *
*                                   *
*  A,B,D,E  DEFINIEREN EINEN BILD-  *
*  SCHIRMAUSSCHNITT,DER C-MAL GE-   *
*  SCROLLT WIRD.                    *
*                                   *
*  Ø<C<25  :NACH UNTEN SCROLLEN     *
*  -25<C<Ø :NACH OBEN SCROLLEN      *
*  A :ERSTE ZEILE                   *
*  B :LETZTE ZEILE                  *
*  C :ANZAHL DES SCROLLENS          *
*  OPTION:                          *
*  D :ERSTE SPALTE     DEFAULT: 1   *
*  E :LETZTE SPALTE    DEFAULT:32   *
*                                   *
*  EXT.BASIC     3Ø6 BYTES          *
*                                   *
*  Ø9.Ø7.1985 BY HEINRICH ACKER     *
*************************************
       IDT  'XSCROLLO'
       DEF  SCROLL
VMBW   EQU  >2Ø24        *
VMBR   EQU  >2Ø2C        *
GPLWS  EQU  >83EØ        *
STATUS EQU  >837C        *
NUMREF EQU  >2ØØC        *
FAC    EQU  >834A        *
XMLLNK EQU  >2Ø18        *
CFI    EQU  >12B8        *
ERR    EQU  >2Ø34        *
ERRSYN EQU  >Ø3ØØ        * SYNTAX ERROR
ERRBV  EQU  >1EØØ        * BAD VALUE
MYWS   BSS  52           *
VDPADR EQU  MYWS+16      * R8
BUFFER EQU  MYWS+18      * AB R9  BUFFER FUER BILDSCHIRMZEILE
SPACE  DATA >8Ø8Ø,>8Ø8Ø,>8Ø8Ø,>8Ø8Ø
       DATA >8Ø8Ø,>8Ø8Ø,>8Ø8Ø,>8Ø8Ø
       DATA >8Ø8Ø,>8Ø8Ø,>8Ø8Ø,>8Ø8Ø
       DATA >8Ø8Ø,>8Ø8Ø,>8Ø8Ø,>8Ø8Ø
*********************************
SCROLL LWPI MYWS
       CLR  RØ
       MOVB @>8312,R1    * >8312 ENTHAELT ANZAHL DER PARAMETER
       SRL  R1,8
       LI   R2,14
       CI   R1,5
       JEQ  NUREF        * 5 PARAMETER
       LI   R2,18
       LI   R7,32        * DEFAULT
       LI   R8,1         * DEFAULT
       CI   R1,3
       JEQ  NUREF        * 3 PARAMETER
       LI   RØ,ERRSYN
       BLWP @ERR         * 'SYNTAX ERROR'
NUREF  BLWP @NUMREF      * BRINGT PARAMETER (SPEZ.IN R1) INS FAC
       BLWP @XMLLNK      * WANDELT FAC IN INTEGER UM
       DATA CFI
       MOV  @FAC,@MYWS(R2)
       JEQ  ERBV         * FALLS IRGENDEIN PARAMETER=Ø --> ERBV
       INCT R2
       DEC  R1
       JNE  NUREF        * NAECHSTER PARAMETER
***  R7:E , R8:D , R9:C , R1Ø:B , R11:A  ***************
       C    R8,R7
       JH   ERBV         * FALLS D>E --> ERBV
       CI   R7,32
       JH   ERBV         * FALLS E>32 --> ERBV
       DEC  R8           * R8 :D-1
       C    R11,R1Ø
       JHE  ERBV         * FALLS A>=B --> ERBV
       CI   R1Ø,24
       JH   ERBV         * FALLS B>24 --> ERBV
       DEC  R1Ø          * R1Ø :B-1
       DEC  R11          * R11 :A-1
       LI   R3,32        * R3 :32
       CI   R9,-24
       JHE  CNEG         * FALLS -24<=C<Ø --> CNEG
```

```
        CI    R9,24
        JLE   CPOS          * FALLS Ø<C<=24 --> CPOS
ERBV    LI    RØ,ERRBV      * SONST
        BLWP  @ERR          * 'BAD VALUE'
CNEG    MOV   R11,R4        * R4 :A-1
        MPY   R3,R4         * R5 :(A-1)*32
        NEG   R9            * R9 :ABS(C)
        JMP   LAB1          * --> LAB1
CPOS    MOV   R1Ø,R4        * R4 :B-1
        MPY   R3,R4         * R5 :(B-1)*32
        NEG   R3            * R3 :-32
LAB1    MOV   R1Ø,R4        * R4 :B-1
        S     R11,R4        * R4 :(B-1)-(A-1)=B-A
        MOV   R4,R6
        INC   R6            * R6 :B-A+1
        C     R9,R6
        JH    LAB2          * FALLS ABS(C)>B-A+1 --> LAB2
        MOV   R9,R6         * R6 :ABS(C)
LAB2    A     R8,R5         * R5 :VDP-ADR. 1.ZEILE/1.SPALTE DES DEF.SCHIRMES
        MOV   R7,R2         * R2 :E
        S     R8,R2         * R2 :E-(D-1) = ANZAHL DER ZEICHEN/ZEILE
LOOP1   LI    R1,BUFFER
        MOV   R5,@VDPADR
        MOV   R4,R7
LOOP2   MOV   @VDPADR,RØ    * LOOP2 SCROLLT ZEILENWEISE RUNTER/HOCH.
        A     R3,RØ         * DIES WIRD DURCH LOOP1 ABS(C)-MAL GETAN.
        BLWP  @VMBR
        MOV   @VDPADR,RØ
        BLWP  @VMBW
        A     R3,@VDPADR
        DEC   R7
        JNE   LOOP2
        MOV   @VDPADR,RØ
        LI    R1,SPACE
        BLWP  @VMBW
        DEC   R6
        JNE   LOOP1
        CLR   RØ
        MOVB  RØ,@STATUS
        LWPI  GPLWS
        B     @>ØØ7Ø        * RUECKSPRUNG INS XBAS.
        END
```





LIFE

Das Assembler-Programm ist für das Minimem geschrieben und wird mit der Option RUN unter dem Namen LIFE aufgerufen. Danach wird man gefragt, ob man nach jeder Generation einen Stop haben will. Für die Elementeneingabe dienen die Cursortasten E, S, D und X. Ein Element wird mit der Taste 1 gesetzt und mit der Taste 0 gelöscht. Eine Eingabe beendet man mit Enter. Diese Eingabe bzw. Änderung kann auch während des Ablaufes durch die Quit-Taste aufgerufen werden; die =-Taste dient als Start-/Stoptaste. Das Programm wird mit Enter verlassen. Sterben alle Elemente aus, so stoppt das Programm automatisch.

Frank Rieger

FAST COPY

Dieses Programm läuft unter Editor/Assembler mit 2 Laufwerken. Die Copydiskette muß dabei vorher mit dem Diskmanager initialisiert werden. Das Programm erstellt eine Sektor-Kopie der Masterdiskette, dabei wird ein eventuell vorhandener Inhalt auf der Copydiskette völlig gelöscht. Wer keine zwei Laufwerke besitzt, kann sich das Programm einfach umändern (an den bezeichneten Stellen muß eine entsprechende Meldung mit Tastenabfrage eingefügt werden). Wesentlicher Punkt dieses Programms ist das gezielte Lesen und Schreiben einzelner Sektoren. Um dieses einfacher in andere Programme einbauen zu können, wurde diese Routine als Unterprogramm ausgeführt. Die Wirkungsweise des Programms selber ergibt sich aus den Kommentaren, so daß dazu eigentlich keine weiteren Ausführungen nötig sind, außer der, daß dieses Programm durch eine "Indiskretion" eines Händlers schon länger auf dem „schwarzen" Markt im Umlauf ist und hier nun alle diejenigen in den „Genuß" dessen kommen sollen, die nicht Zugang zu solchen Tauschkanälen haben.

Heiner Martin



```
*********************************
*                               *
*          LIFE                 *
*                               *
*********************************

        DEF   LIFE

VMBW    EQU   >602B
VSBW    EQU   >6024
VSBR    EQU   >602C
KSCAN   EQU   >6020

TMODE   EQU   >B374
ASCII   EQU   >B375
GPLSTA  EQU   >B37C

ST      BSS   2
BU      BSS   8
B1      BSS   4B4
H1      BSS   440

LIFE    JMP   W1
        DATA  100
T0      TEXT  'STOP'
T1      TEXT  'GENERATION: '
T2      TEXT  'ELEMENTE: '
T3      DATA  >003C,>4242,>4242,>3C00
        DATA  >FFFF,>FFFF,>FFFF,>FFFF
T4      TEXT  'ENTER=ENDE'
T5      TEXT  '  =  =STOP'
T6      TEXT  'QUIT=AENDERN'
T7      TEXT  'AUTOMATISCHER STOP ? (J/N)'

W1      MOV   11,12
        LI    0,>0908
        LI    1,T3
        LI    2,>0010
        BLWP  @VMBW
        CLR   @ST
        LI    0,B1
        LI    1,>00F2
        CLR   2
M0      MOV   2,*0+
        DEC   1
        JGT   M0
        LI    0,>02FF
        LI    1,>2000
L0      BLWP  @VSBW
        DEC   0
        JGT   L0
        JEQ   L0
        LI    0,>0182
        LI    1,T7
        LI    2,>001A
        BLWP  @VMBW
A9      CLR   @TMODE
        BLWP  @KSCAN
        MOVB  @ASCII,0
        SRL   0,8
        CLR   @GPLSTA
        CI    0,>004E
        JEQ   AB
        CI    0,>004A
        JNE   A9
        SETO  @ST
AB      LI    0,>0182
        LI    1,>2000
        LI    2,>001A
A7      BLWP  @VSBW
        INC   0
        DEC   2
        JGT   A7
        LI    0,>004A
        LI    1,>2200
        LI    2,>0016
C0      BLWP  @VSBW
        AI    0,>02A0
        BLWP  @VSBW
        AI    0,>FD61
        DEC   2
        JGT   C0
        LI    0,>006A
        LI    2,>0014
C1      BLWP  @VSBW
        AI    0,>0015
        BLWP  @VSBW
        AI    0,>000B
        DEC   2
        JGT   C1
        LI    0,>0062
        LI    2,>000A
        LI    3,T4
        BL    @PR
        LI    0,>0064
        LI    2,>000A
        LI    3,T5
        BL    @PR
        LI    0,>00B6
        LI    2,>000C
        LI    3,T6
        BL    @PR
        BL    @E1
        JMP   W6
E1      LI    0,>006B
        MOV   0,2
        MOV   0,3
        AI    3,>0013
        LI    4,B1
        AI    4,>0017
MB      BLWP  @VSBR
        MOV   1,5
        LI    1,>1E00
        BLWP  @VSBW
M1      CLR   @TMODE
        BLWP  @KSCAN
        MOVB  @GPLSTA,6
        SLA   6,3
        JNC   M1
        MOVB  @ASCII,6
        SRL   6,8
        MOV   5,1
        BLWP  @VSBW
        CLR   @GPLSTA
        CI    6,>0053
        JEQ   M2
        CI    6,>0044
        JEQ   M3
        CI    6,>0045
        JEQ   M4
        CI    6,>0058
        JEQ   M5
        CI    6,>0031
        JEQ   M6
        CI    6,>0030
        JEQ   M7
        CI    6,>000D
        JNE   M8
        B     *11
M2      C     0,2
        JEQ   M8
        DEC   0
        DEC   4
        JMP   MB
M3      C     0,3
        JEQ   MB
        INC   0
        INC   4
        JMP   M8
M4      CI    0,>00BB
        JLT   MB
        AI    0,>FFE0
        AI    4,>FFEA
        AI    2,>FFE0
        AI    3,>FFE0
        JMP   M8
M5      CI    0,>02BE
        JGT   MB
        AI    0,>0020
        AI    2,>0020
        AI    3,>0020
        AI    4,>0016
        JMP   M8
M6      LI    7,>0100
        MOVB  7,*4
        LI    1,>2100
        JMP   M9
M7      CLR   7
        MOVB  7,*4
        LI    1,>2000
M9      BLWP  @VSBW
        JMP   M8
W6      LI    1,T1
        LI    0,>0002
        LI    2,>000C
        BLWP  @VMBW
        LI    1,T2
        LI    0,>0024
        LI    2,>000A
        BLWP  @VMBW
        CLR   15
L7      LI    0,B1
        MOV   0,1
        MOV   0,2
        AI    1,>0016
        AI    2,>002C
        LI    4,>0014
        LI    5,H1
        CLR   14
L4      CLR   13
        MOVB  13,*5+
        LI    3,>0014
L3      MOVB  *0+,7
        AB    *0+,7
        AB    *0,7
        DEC   0
        AB    *1+,7
        MOVB  *1,13
        AB    *1+,7
        AB    *1,7
        DEC   1
        AB    *2+,7
        AB    *2+,7
        AB    *2,7
        DEC   2
        SRL   7,B
        CI    7,>0003
        JEQ   W2
        SRL   13,8
        S     13,7
        CI    7,>0003
        JEQ   W2
        CLR   7
        MOVB  7,*5+
        JMP   W3
W2      LI    7,>0100
        MOVB  7,*5+
        INC   14
W3      DEC   3
        JGT   L3
        INCT  0
        INCT  1
        INCT  2
        CLR   7
        MOVB  7,*5+
        DEC   4
        JGT   L4
        INC   15
        MOV   15,3
        LI    0,>000E
        BL    @ZA
        LI    0,>006B
        MOV   0,2
        LI    3,H1
        INC   3
```

```
         LI    5,>0014
         LI    6,B1
         AI    6,>0017
L6       LI    4,>0014
L5       MOVB *3+,1
         MOVB 1,*6+
         AI    1,>2000
         BLWP @VSBW
         INC   0
         DEC   4
         JGT   L5
         AI    2,>0020
         MOV   2,0
         INCT 3
         INCT 6
         DEC   5
         JGT   L6
         MOV   14,3
         LI    0,>002E
         BL    @ZA
         CI    14,>0000
         JEQ   B0
         MOV   @ST,0
         JNE   B0
         CLR   @TMODE
         BLWP @KSCAN
         MOVB @ASCII,0
         SRL   0,8
         CLR   @GPLSTA
         CI    0,>000D
         JEQ   EN
         CI    0,>0005
         JNE   A1
         BL    @E1
         B     @L7
A1       CI    0,>003D
         JEQ   B8
         B     @L7
B8       LI    0,>7FFF
B9       DEC   0
         JGT   B9
B0       LI    0,>001C
         LI    1,T0
         LI    2,>0004
         BLWP @VMBW
Z9       CLR   @TMODE
         BLWP @KSCAN
         MOVB @ASCII,13
         SRL   13,8
         CLR   @GPLSTA
         CI    13,>000D
         JEQ   EN
         CI    13,>003D
         JEQ   Z0
         CI    13,>0005
         JNE   Z9
Z0       LI    0,>001C
         LI    1,>70BE          * REG 3
         LI    2,>0004
```

```
         LI    3,>2020
         MOV   3,4
         BLWP @VMBW
         CI    13,>0005
         JNE   Z1
         BL    @E1
Z1       B     @L7
EN       B     *12
ZA       LI    4,BU
         LI    5,>2020
         MOV   5,*4+
         MOV   5,*4
         LI    1,BU
         LI    2,>0004
         BLWP @VMBW
         LI    4,>000A
         LI    5,BU
         AI    5,>0007
         LI    6,>3000
         CLR   7
P2       CLR   2
         DIV   4,2
         SLA   3,8
         AB    6,3
         MOVB 3,*5
         INC   7
         DEC   5
         MOV   2,3
         JNE   P2
         MOV   5,1
         INC   1
         MOV   7,2
         BLWP @VMBW
         B     *11
PR       MOVB *3+,1
         BLWP @VSBW
         AI    0,>0020
         DEC   2
         JGT   PR
         B     *11

         END
```

```
*************************************
*                                   *
*            FAST-COPY              *
*                                   *
*  SEKTORKOPIERPROGRAMM FUER 2 DISKS *
*                                   *
*         H.Martin  7/84            *
*                                   *
*                                   *
*************************************

      DEF   START

      REF   VMBW,VMBR,VSBW,VSBR
      REF   VWTR,GPLLNK,DSRLNK
      REF   KSCAN

      AORG >2800         * FUER ASSEMBLER WICHTIG, DA BUFFER
*                          AB >A000
MYWS  BSS   32

START LWPI MYWS          * EIGENER WORKSPACE
      BL    @CLRSCR      * BILDSCHIRM LOESCHEN
      LI    0,40         * TEXTE AUF DEN BILDSCHIRM
      LI    1,TEXT1
      LI    2,16
      BLWP @VMBW
      LI    0,130
      LI    1,TEXT2
      LI    2,24
      BLWP @VMBW
      LI    0,194
      LI    1,TEXT3
      LI    2,22
      BLWP @VMBW
      LI    0,258
      LI    1,TEXT4
      LI    2,25
      BLWP @VMBW
      LI    0,324
      LI    1,TEXT9
      LI    2,12
      BLWP @VMBW
      BL    @ACTON       * FERTIG, TON
COPY1 LIMI >0002         * WEGEN TON INTERRUPT ZULASSEN
      LIMI >0000
      CLR   0
      MOVB 0,@>837C      * GPL-STATUSBYTE LOESCHEN
      CLR   @>8374       * TASTATURMODUS 0
      BLWP @KSCAN        * TASTATURABFRAGE
      MOVB @>837C,0      * KEINE TASTE GEDRUECKT
      JEQ   COPY1        * DANN VON VORNE
      MOVB @>8375,8
      SRL   8,8
      CI    8,>000D      * ENTER TASTE?
      JEQ   COPY2
      BL    @BADTON      * FALSCHE TASTE, DANN TON
      JMP   COPY1        * VON VORNE
```

```
COPY2  LI   Ø,64          * BILDSCHIRM TEILWEISE LOESCHEN
       CLR  13            * FLAG
       BL   @CLRSC1
       LI   Ø,262         * TEXT 'IN ARBEIT'
       LI   1,TEXT5
       LI   2,9
       BLWP @VMBW
       CLR  4             * SECTOR Ø
       LI   5,>ØCØØ       * VDP-ADRESSE DES BUFFERS
       LI   6,>Ø1ØØ       * NUMMER DES LAUFWERKES
       BL   @RDSEC        * SEKTOR LESEN
       JEQ  ERROR         * BEI FEHLER
       MOVB @>B35Ø,1Ø     * EVENTUELLER FEHLER ABFRAGEN
       JNE  ERROR         * SPRUNG BEI FEHLER
       LI   Ø,>ØCØA
       LI   1,MYWS+14     * ANZAHL SEKTOREN AUS SEKTOR Ø
       LI   2,2           * IN R7 (MYWS+14) LESEN
       BLWP @VMBR
       DEC  7             * MINUS 1, DA SEKTOREN AB Ø
       LI   4,119         * 12Ø SEKTOREN PASSEN INS RAM
       MOV  4,9
COPYB  LI   B,>DØØØ       * BUFFER

* HIER GGF. MELDUNG "MASTERDISK EINLEGEN"
* UND ABFRAGE NACH TASTENDRUCK EINFUEGEN

       LI   6,>Ø1ØØ       * LAUFWERK NUMMER
COPY5  LI   5,>36ØØ       * BUFFER IM VDP
COPY3  LI   Ø,276         * SEKTORNUMMER ANZEIGEN
       MOV  4,15
       BL   @DISPL
       BL   @RDSEC        * SEKTOR LESEN
       JEQ  ERROR         * ERROR ABFRAGEN
       MOVB @>B35Ø,1Ø
       JNE  ERROR
       DEC  9             * MINUS 1
       DEC  4             * MINUS 1
       AI   5,>FFØØ       * VDP-BUFFER MINUS >1ØØ
       CI   5,>ØEØØ       * VDP VOLL?
       JNE  COPY3         * NEIN, WEITER
       MOV  9,9           * SCHON 12Ø SEKTOREN?
       JLT  COPY4         * JA, SPRUNG
       MOV  8,1           * SKTOREN AUS DEM VDP IN DIE RAM-
       LI   Ø,>ØFØØ       * ERWEITERUNG
       LI   2,>2BØØ
       BLWP @VMBR
       LI   8,>AØØØ       * NEUER POINTER ZUM BUFFER IM RAM
       JMP  COPY5         * DA CAPO

* HIER GGF. MELDUNG "COPYDISK EINLEGEN"
* UND ABFRAGE NACH TASTENDRUCK EINFUEGEN
* DANN WEITER UNTEN LAUFWERKNUMMER 1 EINSETZEN

COPY4  LI   8,>AØØØ       * POINTER ZUM RAM
       INC  4             * NAECHSTER SEKTOR
       INC  9             * DITO
COPY7  LI   5,>ØFØØ       * WIR BEGINNEN RUECKWAERTS
       LI   6,>Ø2ØØ       * LAUFWERKNUMMER 2
COPY6  LI   Ø,276         * SEKTORNUMMER ANZEIGEN
       MOV  4,15
       BL   @DISPL
       BL   @WRSEC        * SCHREIBE SEKTOR
       JEQ  ERROR
       MOVB @>835Ø,1Ø     * ERRORABFRAGE
       JNE  ERROR
       INC  4             * NAECHSTER SEKTOR
       INC  9             * NAECHSTER SEKTOR
       AI   5,>Ø1ØØ       * VDP-BUFFER NAECHSTER SEKTOR
       CI   5,>37ØØ       * ENDE ERREICHT?
       JNE  COPY6         * NEIN, WEITER
       MOV  B,1           * NAECHSTE SEKTOREN AUS DEM
       LI   Ø,>ØFØØ       * RAM-BUFFER HOLEN
       LI   2,>2BØØ
       BLWP @VMBW
       LI   B,>DØØØ       * POINTER ZUM NAECHSTEN BLOCK
       CI   9,12Ø         * SCHON ALLE?
       JNE  COPY7         * NEIN, WEITER
       DEC  9             * WAREN NUR 12Ø
       AI   4,119         * GESAMTANZAHL
       C    4,7           * REST FESTSTELLEN
       JLT  COPYB         * GROESSER ALS 12Ø, DANN WEITER
       MOV  7,4           * REST DER SEKTOREN
       MOV  13,13         * FLAG PRUEFEN
       JNE  ENDE          * WAR SCHON EINMAL REST, DANN ENDE
       SETO 13            * FLAG SETZEN FUER LETZTEN DURCHLAUF
       JMP  COPY8

ERROR  LI   Ø,196         * TEXT 'ERROR'
       LI   1,TEXT6
       LI   2,11
       BLWP @VMBW
       MOVB 1Ø,1Ø         * SEKTOR-ERROR
       JNE  ERRO1
       MOVB 1,1Ø          * ERRORCODE
ERRO1  AI   Ø,14          * ERRORCODE ANZEIGEN
       MOV  1Ø,1
       SRL  1,12          * HOECHSTWERTIGSTES NYBBLE
       SLA  1,8
       AI   1,>3Ø3Ø       * IN ASCII
       SLA  1Ø,4          * NIEDRIGWERTIGES NYBBLE
       SRL  1Ø,12
       A    1Ø,1
       BLWP @VSBW         * ANZEIGEN
       INC  Ø
       SWPB 1
       BLWP @VSBW         * KOMPLETT
       BL   @BADTON       * TON UND ENDE
       JMP  ENDE1

ENDE   LI   Ø,26Ø         * TEXT ANWEISUNG
       LI   1,TEXT7
       LI   2,23
       BLWP @VMBW
       BL   @ACTON        * TON
ENDE1  LI   Ø,324
       LI   1,TEXT8       * TEXT WEITER
       LI   2,17
       BLWP @VMBW
```

```
ENDE2  LIMI >ØØØ2         * INTERRUPT WEGEN TON
       LIMI >ØØØØ
       CLR  Ø             * TASTATURABFRAGE WIE OBEN
       MOVB Ø,@>837C
       CLR  @>8374
       BLWP @KSCAN
       MOVB @>837C,Ø
       JEQ  ENDE2
       MOVB @>8375,1
       SRL  1,8
       CI   1,>ØØØC       * TASTE PROCEED?
       JEQ  ENDE3         * JA, WEITER
       BL   @BADTON       * FALSCHE TASTE, TON
       JMP  ENDE2
ENDE3  B    @START


TEXT9  TEXT 'By H. Martin'
       EVEN

**************************
* ACCEPT TONE AUSGEBEN

ACTON  CLR  14
       MOVB 14,@>837C     * GPL-STATUS LOESCHEN
       BLWP @GPLLNK       * TON UEBER GPLLNK AUFRUFEN
       DATA >ØØ34
       RT

**************************
* BAD ACCEPT TON AUSGEBEN

BADTON CLR  14
       MOVB 14,@>837C     * GPL-STATUS LOESCHEN
       BLWP @GPLLNK       * TON UEBER GPLLNK AUFRUFEN
       DATA >ØØ36
       RT                 * RETURN 11

**************************
* SCREEN LOESCHEN
* HINTERHER:

CLRSCR CLR  Ø             * SCREEN LOESCHEN
CLRSC1 LI   2,>Ø3ØØ
       LI   1,>2Ø2Ø       * CLRSC1 FUER TEILWEISES LOESCHEN
CLRSC2 BLWP @VSBW         * RØ=ADRESSE, R2=ANZAHL
       INC  Ø             * NAECHSTE BILDSCHIRMADRESSE
       C    Ø,2           * ENDE ERREICHT?
       JNE  CLRSC2        * NEIN, WEITER
       RT                 * RETURN 11

**************************
* READ AND WRITE SECTOR

RDSEC  LI   1,>Ø1ØØ       * KENNWERT FUER READ
       JMP  SEC1
WRSEC  CLR  1             * KENNWERT FUER WRITE
SEC1   MOVB 1,@>834D      * KENNWERT FUER UNTERPROGRAMM SEKTOR
       MOV  4,@>835Ø      * SEKTORNUMMER
```

```
       MOV  5,@>834E      * VDP-ADRESSE FUER BUFFER
       MOVB 6,@>834C      * LAUFWERKNUMMER
       LI   Ø,>Ø48Ø       * PSEUDO-PAB SCHREIBEN AB >Ø48Ø IM VDP
       MOV  Ø,@>8356      * POINTER FUER DSRLNK
       LI   3,>Ø11Ø       * SUBPROGRAMM NAME >1Ø, LAENGE >Ø1
       LI   2,2
       LI   1,MYWS+6      * R3
       BLWP @VMBW         * "PAB" INS VDP-RAM
       BLWP @DSRLNK       * UND AUFRUFEN
       DATA 1Ø            * FUER SUBPROGRAMME
       B    *11

*****************************
* DISPL

DTEN   DATA 1Ø            * FUER DEZIMALWERT

DISPL  CLR  14            * VORBEREITEN FUER DIVISION
       DIV  @DTEN,14      * R14 GETEILT DURCH 1Ø
       AI   15,>ØØ3Ø      * REST IN ASCII VERWANDELN
       SWPB 15
       MOVB 15,1
       BLWP @VSBW         * AUF DEN BILDSCHIRM
       DEC  Ø             * NAECHSTE VDP-ADRESSE (STELLE)
       MOV  14,15         * NOCH ETWAS ZUM TEILEN DA?
       JNE  DISPL         * JA, VON VORNE
       LI   1,>2Ø2Ø       * STELLE DAVOR NOCH LOESCHEN
       BLWP @VSBW
       B    *11

TEXT1  TEXT '**  FASTCOPY  **'
TEXT2  TEXT 'MASTERDISK IN LAUFWERK 1'
TEXT3  TEXT 'COPYDISK IN LAUFWERK 2'
TEXT4  TEXT 'DANN ENTER-TASTE DRUECKEN'
TEXT5  TEXT 'IN ARBEIT'
TEXT6  TEXT 'ERROR-CODE:'
TEXT7  TEXT 'ANWEISUNG ABGESCHLOSSEN'
TEXT8  TEXT 'WEITER MIT PROC'D'

       EVEN

       END
```

GPLLNK

GPLLNK FÜR EXTENDED BASIC

Es hat in der Vergangenheit schon mehrere Vorschläge für ein GPLLNK unter Extended Basic gegeben. Entweder waren diese Programme aber sehr lang, oder sie funktionierten nicht richtig. Sogar die von Texas Instruments USA verschickte Version hatte einen Fehler, man konnte nicht zweimal hintereinander eine der beiden Ton-Routinen aufrufen.
Die hier vorgestellte, sehr kurze Version basiert durchaus auf dem Vorschlag von Texas Instruments, jedoch wurde der Fehler beseitigt und dazu noch das Programm gekürzt. Verwendet wird die Routine genauso wie sonst das GPLLNK im Editor/Assembler-Modul bzw. Mini-Memory, d.h. also nach dem BLWP-Aufruf kommt ein folgendes DATA mit der Grom-Adresse der aufzurufenden Routine. Beim Assemblieren wird es einfach an das Programm, welches diese Routine benötigt, hinten angehängt. Anmerken muß ich hier noch, daß dieses GPLLNK ausschließlich unter Extended Basic funktioniert.

Heiner Martin

```
*************************************
*                                   *
*     GPLLNK fuer EXTENDED BASIC    *
*                                   *
*        3.7.84 H. Martin           *
*                                   *
*************************************

*       EQUATES

UTLWS  EQU  >2038         * UTILITIE WORKSPACE
SUBST  EQU  >8373         * SUBSTACK POINTER
GRMRA  EQU  >9802         * GROM READ ADRESSE
GPWS   EQU  >83E0         * GPL-WORKSPACE

*       EINSPRUNGVEKTOREN

GPLLNK DATA UTLWS
       DATA GPLLN1

*       PROGRAMM

GPLLN1 MOVB @GRMRA,0      * GROM ADRESSE LESEN
       SWPB 0
       MOVB @GRMRA,0
       SWPB 0
       AI   0,-3          * ZURUECK ZUM XML BEFEHL VON CALL LINK
       MOVB @SUBST,1      * GPL-SUBSTACK POINTER
       SRL  1,8
       AI   1,>8300       * KOMPLETTE ADRESSE DES SUBSTACK
       INCT 1             * XML ADRESSE FUER RETURN AUF
       MOV  0,*1          * DEN STACK LADEN
       SWPB 1
       MOVB 1,@SUBST      * NEUER SUBSTACK-POINTER
       LI   3,>2000
       MOV  *3,2          * RETTE XML LINK
       LI   0,GPLLN2      * NEUES XML LINK
       MOV  0,*3          * AUF DEN XML-POINTER
       MOV  *14+,@>83EC   * GPL-ADRESSE (DATA) IN R6 GPLWS
       LWPI GPWS          * GPL-WORKSPACE LADEN
       B    @>0060        * ROUTINE AUFRUFEN MIT RESET GPL
GPLLN2 LWPI UTLWS
       MOV  2,*3          * ALTES XML LINK WIEDERHERSTELLEN
       RTWP               * UND ZURUECK
```

BIT MAP MODE

Das vorliegende Bit-Map-Mode-Programm erlaubt es Ihnen, mittels 16 Befehlen Grafiken im Bit-Map-Mode, dem höchsten Auflösungs-Mode auf dem TI 99/4A, zu erzeugen, abzuspeichern, zu laden, auszudrucken und sich auf dem Bildschirm anzuschauen.

Bevor ich auf die einzelnen Befehle eingehe, möchte ich noch ein paar Anmerkungen zu meinem Programm loslassen.

Die Speichererweiterung ist trotz Minimem's Voraussetzung. Die Grafiken werden blind erzeugt, d.h. die Grafik wird in einem speziellen Speicherbereich in der Speichererweiterung erstellt und kann mittels eines Befehles für eine bestimmte Zeit auf dem Bildschirm gezeigt werden. Danach kehrt das Maschinenprogramm zum Titelbildschirm zurück. Das Basic-Programm zur Erzeugung der Grafik ist somit gelöscht, dies ist auch schon der Nachteil des Bit-Map-Mode's. Sie können sich die Grafik allerdings immer wieder von Basic her anschauen und verändern, da diese in der Speichererweiterung erhalten bleibt. Ein Diskettenlaufwerk ist wünschenswert, da das Programm die Abspeicherung der Grafiken nur auf Diskette unterstützt. Auch eine Hardcopy für den Seikosha GP 100 A ist vorhanden.
Alle Befehle werden mittels CALL LINK und eines Programmnamens aufgerufen. Zum Teil müssen Parameter mit übergeben werden. Es sind nur Strings, Stringvariablen, Zahlen und numerische Variablen erlaubt.
Falsche Parameter führen zu einer Fehlermeldung und Programmabbruch.

Befehlssatz:

CALL LINK ("CLEAR") —>
Dieser Befehl löscht die Grafik. Er sollte als erstes in Ihrem Programm stehen.

CALL LINK ("SETPIX", Y,X,(bis zu 7 mal))—>
Dieser Befehl setzt an die Koordinate X,Y einen Punkt. Es können bis zu sieben Punkte gleichzeitig gesetzt werden.

CALL LINK ("CLRPIX", Y,X,(bis zu 7 mal))—>
Dieser Befehl löscht einen Punkt mit der Koordinate X,Y. Es können bis zu 7 Punkte gleichzeitig gelöscht werden.

CALL LINK ("WRITE", A, Zeile, Spalte, STRING)—>
Dieser Befehl ermöglicht es, Texte in die Grafik zu schreiben. Wenn A gleich 1 ist, dann kann man den Text in die Grafik hineinkopieren und wenn A gleich 0 ist, dann wird die Grafik unter dem Text gelöscht. Die Textposition wird durch Zeile und Spalte bestimmt. Es sind Strings mit einer Länge von bis zu 32 Zeichen erlaubt. Zur Verfügung ste-

Bitte weiter auf S. 61

```
*       UNL
* BIT-MAP-GRAFIC
* ==============
*
* (c) by Bernd Bertling, Neu-Crengeldanzstra~e 2
*                        4600 Dortmund 72
*
*
* Programmidentifizierung
*
        IDT  'BITMGRFC'
*
* Bei Erstellung eines Objektcodes f}r Editor/Assembler m}~en folgende
* Refferenzen statt der unten genannten Equates geschrieben werden.
*
* Refferenzen f}r Editor/Assembler
*
*       REF  VSBW,VSBR,VMBW,VMBR,VWTR,ERR
*       REF  XMLLNK,NUMREF,STRREF,DSRLNK,GPLLNK
*
        AORG >A000  Absolute-Code f}r Minimem
*
* Programmnamen Definition
*
        DEF  CLEAR,SETPIX,CLRPIX,ZEIG,WRITE,CHAR,COLOR,SAVE
        DEF  OLD,HCOPY,KREIS,CKREIS,LINE,CLINE,WLINE,CWLINE
*
* Equates f}r Minimem
*
VSBR   EQU  >602C     Ein Byte aus dem VDP-RAM lesen
VSBW   EQU  >6024     Ein Byte in's VDP-RAM schreiben
VMBR   EQU  >6030     Mehrere Bytes aus dem VDP-RAM lesen
VMBW   EQU  >6028     Mehrere Bytes in's VDP-RAM schreiben
VWTR   EQU  >6034     VDP-Register ver{ndern
XMLLNK EQU  >601C     Verbindung zu ROM-Routinen
NUMREF EQU  >6044     Numerische Parameter: Basic --> Assembler
STRREF EQU  >604C     Strings: Basic --> Assembler
ERR    EQU  >6050     Fehlermeldung: Assembler --> Basic
DSRLNK EQU  >6038     Verbindung zur DSR-Routine
GPLLNK EQU  >6018     Verbindung zu GROM-Routinen
*
* Sonstige Equates
*
SIN    EQU  >002E     GPL-Routine --> Sinus-Funktion
COS    EQU  >002C     GPL-Routine --> Cosinus-Funktion
RES    EQU  >0038     Platz im VDP-RAM reservieren
FADD   EQU  >0600     ROM-Routine --> Gleitkomma Addition
FMULT  EQU  >0800     ROM-Routine --> Gleitkomma Multiplikation
FDIV   EQU  >0900     ROM-Routine --> Gleitkomma Division
CFI    EQU  >1200     Gleitkomma --> Integer
* Folgender Wert mu~ bei der Editor/Assembler-Version von >7200 in
* >2300 ge{ndert werden.
CIF    EQU  >7200     Integer --> Gleitkomma
ERR22  EQU  >1600     'BAD-ARGUMENT' - Fehlermeldung
LNKNUM EQU  >8312     Anzahl der Variablen in CALL LINK
WATCH  EQU  >8379     VDP-Interrupt-Timer
GPLSTA EQU  >837C     Statusregister
FAC    EQU  >834A     Gleitkomma Accu
ARG    EQU  >835C     Gleitkomma Argument
```

```
VCOP1  EQU  >B3D4      Kopieadresse des VDP-Registers 1
GPLWS  EQU  >B3EØ      GPL-Arbeitsbereich
POINT  EQU  >B31C      Pab-Pointer
PNTR   EQU  >B356      DSR-Namenszeiger
*
* Variablen
*
PABBUF BSS  2    Variable f)r Pab-Buffer-Adresse
PAB    BSS  2    Variable f)r Pab-Adresse
PAB5   BSS  2    Adressen-Variable
BACK1  BSS  2    Returnadressen-Buffer f)r Unterprogramme
BACK2  BSS  2    Returnadressen-Buffer f)r Unterprogramme
MYREGI BSS  32   Eigene Arbeitsregister
MYREG2 BSS  32   2.Satz Arbeitsregister
HELP1  BSS  2    Hilfs-Variablen
HELP2  BSS  2      "  -    "
HELP3  BSS  2      "  -    "
STRINØ BSS  B      "  -    "
STRIN1 BSS  B      "  -    "
STRIN2 BSS  B      "  -    "
SATZ   BSS  33   33 Byte Variablen-Buffer
PRMZAL BSS  2    Schleifenz{hler
ANZAHL BSS  2          "
MATRIX BSS  56   Hilfsvariablen-Feld
P      BSS  2    Pixel setzen oder l|schen
BUFADR BSS  2    Buffer f)r aktuelle Buffer-Adresse
PATT   BSS  6144 Pattern-Description-Buffer
FARBE  BSS  6144 Farblisten-Buffer
BUFFER BSS  76B  Standard-ASCII-Buffer
FPBUF1 BSS  B
FPBUF2 BSS  B
XAFP   BSS  B
YAFP   BSS  B
XEFP   BSS  B
YEFP   BSS  B
WINKEL BSS  B    Beinhaltet den aktuellen Winkel in Radiant
*
XA     BSS  2
YA     BSS  2
XE     BSS  2
YE     BSS  2
X      BSS  2    Hilfsvariable
Y      BSS  2          "
RQ     BSS  2          "
*
* Datei-Daten f)r Disketten-Datei
*
PDATA  DATA >ØØØØ,>ØØØØ,>5ØØØ,>ØØØØ,>ØØØØ
*
* Datei-Daten f)r Drucker-Datei
*
PDATA2 DATA >ØØ12,>ØØØØ,>BØØØ,>ØØØØ,>ØØØ6
       TEXT 'PIO.CR'
MYDAT1 BYTE >ØA,>1B,>1Ø,>ØØ,>6B
MYDAT2 BYTE >1B,>1Ø,>ØØ,>EB
MYDAT3 BYTE >ØA,>ØA,>ØA,>ØA,>ØA
*
* VDP-Register
*
```

```
VREG1  DATA >ØØØ2,>Ø1EØ,>Ø2Ø6,>Ø3FF,>Ø4Ø3,>Ø536,>Ø6ØØ,>Ø712  Bitmap-Mode
VREG2  DATA >ØØØØ,>Ø1EØ,>Ø2ØØ,>Ø3ØC,>Ø4ØØ,>Ø5Ø6,>Ø6ØØ,>Ø7Ø7  Grafic-Mode
*
* Konstanten
*
BYTM55 BYTE -55
BYTM4B BYTE -4B
BYTØ   BYTE Ø
BYT1   BYTE 1
BYT2   BYTE 2
WRITEE BYTE 3
BYT15  BYTE 15
BYT16  BYTE 16
BYT32  BYTE 32
BYT5Ø  BYTE 5Ø
BYT12B BYTE 12B
       EVEN
DØ1    DATA 1
DØ2    DATA 2
DØ5    DATA 5
DØB    DATA B
DØ9    DATA 9
D16    DATA 16
D25    DATA 25
D32    DATA 32
DBØ    DATA BØ
D1Ø5   DATA 1Ø5
D12B   DATA 12B
D152   DATA 152
D192   DATA 192
D256   DATA 256
D16257 DATA 16257
DIVISO DATA >4Ø39,>1D39,>5ØØØ,>ØØØØ    gleich 57.2957B (Gleitkommazahl)
*
* Weitere Drucker-Daten
*
ZLDATA DATA Ø,Ø,STRIN2,>ØØØ7
       DATA Ø,256,STRIN1+7,>Ø1Ø6
       DATA 256,512,STRIN1+6,>Ø2Ø5
       DATA 512,76B,STRIN1+5,>Ø3Ø4
       DATA 76B,1Ø24,STRIN1+4,>Ø4Ø3
       DATA 1Ø24,12BØ,STRIN1+3,>Ø5Ø2
       DATA 12BØ,1536,STRIN1+2,>Ø6Ø1
       DATA 1536,1536,STRIN1+1,>Ø7ØØ
       DATA 1792,1792,STRIN2,>ØØØ7
       DATA 1792,2Ø4B,STRIN1+7,>Ø1Ø6
       DATA 2Ø4B,23Ø4,STRIN1+6,>Ø2Ø5
       DATA 23Ø4,256Ø,STRIN1+5,>Ø3Ø4
       DATA 256Ø,2B16,STRIN1+4,>Ø4Ø3
       DATA 2B16,3Ø72,STRIN1+3,>Ø5Ø2
       DATA 3Ø72,332B,STRIN1+2,>Ø6Ø1
       DATA 332B,332B,STRIN1+1,>Ø7ØØ
       DATA 35B4,35B4,STRIN2,>ØØØ7
       DATA 35B4,3B4Ø,STRIN1+7,>Ø1Ø6
       DATA 3B4Ø,4Ø96,STRIN1+6,>Ø2Ø5
       DATA 4Ø96,4352,STRIN1+5,>Ø3Ø4
       DATA 4352,46ØB,STRIN1+4,>Ø4Ø3
       DATA 46ØB,4B64,STRIN1+3,>Ø5Ø2
       DATA 4B64,512Ø,STRIN1+2,>Ø6Ø1
```

```
       DATA 512Ø,512Ø,STRIN1+1,>Ø7ØØ
       DATA 5376,5376,STRIN2,>ØØØ7
       DATA 5376,5632,STRIN1+7,>Ø1Ø6
       DATA 5632,5888,STRIN1+6,>Ø2Ø5
       DATA 5888,5888,STRIN1+5,>Ø3ØØ
*
* Wurzel-Routinen-Daten
*
SQRDA1 DATA Ø,Ø,2,1,6,2,12,3,2Ø,4,3Ø,5
       DATA 42,6,56,7,72,8,9Ø,9,11Ø,1Ø
       DATA 132,11,156,12,182,13,21Ø,14
       DATA 24Ø,15,272,16,3Ø6,17,342,18
       DATA 38Ø,19,42Ø,2Ø,462,21,5Ø6,22
       DATA 552,23,6ØØ,24,65Ø,25,7Ø2,26
       DATA 756,27,812,28,87Ø,29,93Ø,3Ø
       DATA 992,31,1Ø56,32,1122,33,119Ø,34
       DATA 126Ø,35,1332,36,14Ø6,37
       DATA 1482,38,156Ø,39,164Ø,4Ø
       DATA 1722,41,18Ø6,42,1892,43
       DATA 198Ø,44,2Ø7Ø,45,2162,46
       DATA 2256,47,2352,48,245Ø,49
       DATA 255Ø,5Ø,2652,51,2756,52
       DATA 2862,53,297Ø,54,3Ø8Ø,55
       DATA 3192,56,33Ø6,57,3422,58
       DATA 354Ø,59,366Ø,6Ø,3782,61
       DATA 39Ø6,62,4Ø32,63,416Ø,64
       DATA 429Ø,65,4422,66,4556,67
       DATA 4692,68,483Ø,69,497Ø,7Ø
       DATA 5112,71,5256,72,54Ø2,73
       DATA 555Ø,74,57ØØ,75,5852,76
       DATA 6ØØ6,77,6162,78,632Ø,79
       DATA 648Ø,8Ø,6642,81,68Ø6,82
       DATA 6972,83,714Ø,84,731Ø,85
       DATA 7482,86,7656,87,7832,88
       DATA 8Ø1Ø,89,819Ø,9Ø,8372,91
       DATA 8556,92,8742,93,893Ø,94
       DATA 912Ø,95,9312,96,95Ø6,97
       DATA 97Ø2,98,99ØØ,99,1Ø1ØØ,1ØØ
       DATA 1Ø3Ø2,1Ø1,1Ø5Ø6,1Ø2,1Ø712,1Ø3
       DATA 1Ø92Ø,1Ø4,1113Ø,1Ø5,11342,1Ø6
       DATA 11556,1Ø7,11772,1Ø8,1199Ø,1Ø9
       DATA 1221Ø,11Ø,12432,111,12656,112
       DATA 12882,113,1311Ø,114,1334Ø,115
       DATA 13572,116,138Ø6,117,14Ø42,118
       DATA 1428Ø,119,1452Ø,12Ø,14762,121
       DATA 15ØØ6,122,15252,123,155ØØ,124
       DATA 1575Ø,125,16ØØ2,126,16256,127
SQRDA2 DATA 16512,128,1677Ø,129,17Ø3Ø,13Ø
       DATA 17292,131,17556,132,17822,133
       DATA 1809Ø,134,1836Ø,135,18632,136
       DATA 189Ø6,137,19182,138,1946Ø,139
       DATA 1974Ø,14Ø,2ØØ22,141,2Ø3Ø6,142
       DATA 2Ø592,143,2Ø88Ø,144,2117Ø,145
       DATA 21462,146,21756,147,22Ø52,148
       DATA 2235Ø,149,2265Ø,15Ø,22952,151
       DATA 23256,152,23562,153,2387Ø,154
       DATA 2418Ø,155,24492,156,248Ø6,157
       DATA 25122,158,2544Ø,159,2576Ø,16Ø
       DATA 26Ø82,161,264Ø6,162,26732,163
       DATA 27Ø6Ø,164,2739Ø,165,27722,166
       DATA 28Ø56,167,28392,168,2873Ø,169
       DATA 29Ø7Ø,17Ø,29412,171,29756,172
       DATA 3Ø1Ø2,173,3Ø45Ø,174,3Ø8ØØ,175
       DATA 31152,176,315Ø6,177,31862,178
       DATA 3222Ø,179,3258Ø,18Ø,32942,181
       DATA 333Ø6,182,33672,183,34Ø4Ø,184
       DATA 3441Ø,185,34782,186,35156,187
       DATA 35532,188,3591Ø,189,3629Ø,19Ø
       DATA 36672,191,37Ø56,192,37442,193
       DATA 3783Ø,194,3822Ø,195,38612,196
       DATA 39ØØ6,197,394Ø2,198,398ØØ,199
       DATA 4Ø2ØØ,2ØØ,4Ø6Ø2,2Ø1,41ØØ6,2Ø2
       DATA 41412,2Ø3,4182Ø,2Ø4,4223Ø,2Ø5
       DATA 42642,2Ø6,43Ø56,2Ø7,43472,2Ø8
       DATA 4389Ø,2Ø9,4431Ø,21Ø,44732,211
       DATA 45156,212,45582,213,46Ø1Ø,214
       DATA 4644Ø,215,46872,216,473Ø6,217
       DATA 47742,218,4818Ø,219,4862Ø,22Ø
       DATA 49Ø62,221,495Ø6,222,49952,223
       DATA 5Ø4ØØ,224,5Ø85Ø,225,513Ø2,226
       DATA 51756,227,52212,228,5267Ø,229
       DATA 5313Ø,23Ø,53592,231,54Ø56,232
       DATA 54522,233,5499Ø,234,5546Ø,235
       DATA 55932,236,564Ø6,237,56882,238
       DATA 5736Ø,239,5784Ø,24Ø,58322,241
       DATA 588Ø6,242,59292,243,5978Ø,244
       DATA 6Ø27Ø,245,6Ø762,246,61256,247
       DATA 61752,248,6225Ø,249,6275Ø,25Ø
       DATA 63252,251,63756,252,64262,253
       DATA 6477Ø,254,6528Ø,255,65535,256
*
* Copy Directives
*
       COPY "DSK1.@BITMAP1S"   * Unterprogramme 1.Teil
       COPY "DSK1.@BITMAP2S"   * Unterprogramme 2.Teil
       COPY "DSK1.@BITMAP3S"   * Hauptprogramme 1.Teil
       COPY "DSK1.@BITMAP4S"   * Hauptprogramme 2.Teil
*
* Ende des Programms
*
       END  CLEAR   Automatische Buffer-L!schung
```

```
* Segment "@BITMAP1S"
*
* *****************
* *               *
* * Unterprogramme *
* *               *
* *     1.Teil    *
* *               *
* *****************
*
* BL @PIXEL
*
* Diese Routine setzt an einen, durch Koordinaten definierten Punkt, Pixel.
* Die )bergebenen Werte sind:Y-Koordinate ; X-Koordinate . Setzen oder l!schen
* eines Pixels wird durch den Wert in 'P' bestimmt.Steht eine 1 in 'P', so
* wird ein Punkt gesetzt und bei Ø gel!scht.
*
PIXEL  CLR  R1            R1=Ø
       MOVB @ANZAHL,R1    Anzahl der Parameter in CALL LINK nach R1
       SWPB R1            Byte in Wort verwandeln
       MOV  R1,@ANZAHL    Anzahl der Parameter als Wort zur)ck
       MOV  R1,@PRMZAL    Wert kopieren
*
       CI   R1,2          Sind mind. zwei Parameter vorhanden ?
       JLT  ERRT1         Wenn weniger, Fehlermeldung
       CI   R1,14         Sind max. vierzehn Parameter vorhanden ?
       JGT  ERRT1         Wenn mehr, Fehlermeldung
       JMP  PIXEL1        Wenn alles OK, Koordinaten )bernehmen
*
ERRT1  LIMI 2             VDP-Interrupt wieder zulassen
       B    @BARERR       Fehlermeldung ausgeben
*
PIXEL1 LI   R1,1          R1=1
       LI   R6,SATZ       R6 mit Parameter-Buffer-Adresse laden
*
PIXEL2 BL   @TSTGR        Parameter auf Zul(~igkeit pr)fen (Zeile)
       DATA 1,192         Untere Grenze, Obere Grenze
       MOV  @FAC,*R6+     Parameter in Buffer bringen und Adresse erh!hen
       INC  R1            Parameternummer um eins erh!hen
*
       BL   @TSTGR        Parameter auf Zul(~igkeit pr)fen (Spalte)
       DATA 1,256         UG, OG
       MOV  @FAC,*R6+     Parameter in Buffer bringen und Adresse erh!hen
       INC  R1            Parameternummer um eins erh!hen
*
       DECT @ANZAHL       Alle Parameter geholt ?
       JNE  PIXEL2        Wenn nicht, die n(chsten zwei Parameter holen
*
SETTO  MOV  R11,@BACK1
*
       LI   R15,SATZ      R15 mit Parameter-Buffer-Adresse laden
*
PIXEL3 MOV  *R15+,R2      Y-Koordinate )bernehmen (Wort)
       MOV  *R15+,R4      X-Koordinate )bernehmen (Wort)
*
       CB   @BYT1,R7
       JNE  PIXEL4
       C    R2,@DØ1       Y-Koordinate, Untere Grenze
       JLT  PIXELA
       C    R4,@DØ1       X-Koordinate, Untere Grenze
       JLT  PIXELA
       C    R2,@D192      Y-Koordinate, Obere Grenze
       JGT  PIXELA
       C    R4,@D256      X-Koordinate, Obere Grenze
       JGT  PIXELA
*
PIXEL4 DEC  R2            Y-Koordinate=R2-1
       DEC  R4            X-Koordinate=R4-1
       MOV  R2,R6         Y-Koordinate kopieren
       MOV  R4,R14        X-Koordinate kopieren
       SRL  R6,3          R6=R6/8
       MOV  R6,R12        R12=R6
       SLA  R6,3          R6=R6*8
       S    R6,R2         R2=R2-R6
       SLA  R12,8         R12=R12*256
       A    R12,R2        R2=R2+R12
       MOV  R2,RØ         In RØ steht jetzt einen Teil der fertigen Adresse
       SRL  R4,3          R4=R4/8
       SLA  R4,3          R4=R4*8
       S    R4,R14        R14=R14-R4
       A    R4,RØ         In RØ steht die fertige P.D.T.-Buffer-Adresse
       LI   R11,PATT      P.D.T.-Buffer-Adresse in R11 laden
       A    R11,RØ        Basisadresse zum berechneten Wert addieren
       MOV  RØ,R1Ø        Fertige Adresse nach R1Ø kopieren
       CLR  R12           R12=Ø
       MOVB *R1Ø,R12      Byte nach R1 bringen
*
       MOV  R14,RØ        Bitadresse nach RØ, da RØ gleichzeitig Schiebewert ist
       SWPB R12           Byte in Wort verwandeln
       MOV  R12,R9        Grafic-Wert kopieren
       MOV  RØ,RØ         Ist RØ=Ø ?
       JEQ  PIXEL5        Wenn ja, dann braucht nicht geschoben zu werden
       SLA  R9,Ø          R9 um Wert von RØ nach links schieben (Multiplizieren)
PIXEL5 SLA  R9,8          Wert in den sign. Bereich schieben
       SRL  R9,7          Wert um 7 Bits nach rechts (Dividieren)
       CB   @BYT1,R9      Ist das Pixel bereits gesetzt ?
       JNE  PIXEL7        Wenn nicht, dann zu NXT1 springen
*
       CB   @BYT1,@P      Ist das Pixel bereits gesetzt ?
       JEQ  PIXELA        Wenn ja, dann braucht man es nicht setzen
*
       LI   R13,128       R13=128
       MOV  RØ,RØ         Ist RØ=Ø ?
       JEQ  PIXEL6        Wenn ja, dann braucht nicht geschoben zu werden
       SRL  R13,Ø         Sonst um Wert von RØ nach rechts schieben (Dividieren)
PIXEL6 S    R13,R12       R12=R12-R13
       JMP  PIXEL9        N(chste Runde
*
PIXEL7 CB   @BYTØ,@P      Ist das Pixel )berhaupt gesetzt ?
       JEQ  PIXELA        Wenn nicht, dann braucht man es nicht l!schen
*
       LI   R13,128       R13=128
       MOV  RØ,RØ         Ist RØ=Ø ?
       JEQ  PIXEL8        Wenn ja, dann braucht nicht geschoben zu werden
       SRL  R13,Ø         Sonst um Wert von RØ nach rechts schieben (Dividieren)
PIXEL8 A    R13,R12       R12=R12+R13
*
PIXEL9 SWPB R12           Wort in Byte verwandeln
```

NUTZEN SIE UNSEREN BEQUEMEN POSTSERVICE





KOMMT REGELMÄSSIG ZU IHNEN INS HAUS

Finden Sie Ihre TI REVUE nicht am Kiosk? Weil sie schon ausverkauft ist? Oder „Ihr" Kiosk nicht beliefert wurde? Kein Problem! Für ganze 60 DM liefern wir per Post zwölf Hefte ins Haus (Ausland 80 DM). Einfach den Bestellschein auf der nächsten Seite ausschneiden – fotokopieren oder abschreiben, in einen Briefumschlag und ab per Post (Achtung: Porto nicht vergessen). TI REVUE kommt dann pünktlich ins Haus.

WICHTIGE RECHTLICHE GARANTIE!

Sie können diesen Abo-Auftrag binnen einer Woche nach Eingang der Abo-Bestätigung durch den Verlag widerrufen– Postkarte genügt. Ansonsten läuft dieser Auftrag jeweils für zwölf Ausgaben, wenn ihm nicht vier Wochen vor Ablauf widersprochen wird, weiter.

DAS ANGEBOT: KLEINANZEIGEN KOSTENLOS!

Das bietet Ihnen ab sofort die TI-REVUE: KLEINANZEIGEN SIND KOSTENLOS FÜR PRIVATANBIETER! Suchen Sie etwas, haben Sie etwas zu verkaufen, zu tauschen, wollen Sie einen Club gründen? Coupon ausfüllen, auf Postkarte kleben oder in Briefumschlag stecken und abschicken. So einfach geht das. Wollen Sie das Heft nicht zerschneiden, können Sie den Coupon auch fotokopieren. Oder einfach den Anzeigentext uns so schicken, auf Postkarte oder im Brief. Aber bitte mit Druckbuchstaben oder in Schreibmaschinenschrift!

Und: Einschließlich Ihrer Adresse und/oder Telefonnummer sollten acht Zeilen à 28 Anschläge nicht überschritten werden.

ACHTUNG: WICHTIGER HINWEIS!

Wir veröffentlichen nur Kleinanzeigen privater Inserenten, keine gewerblichen Anzeigen. Die kosten pro Millimeter DM 5,– plus Mehrwertsteuer!

Wir versenden für Privat-Inserenten keine Beleg-Exemplare!

Chiffre-Anzeigen sind nicht gestattet! Wir behalten uns vor, Anzeigen, die gegen rechtliche, sittliche oder sonstige Gebote verstoßen, abzulehnen!

Anzeigenabdruck in der Reihenfolge ihres Eingangs, kein Rechtsanspruch auf den Abdruck in der nächsten Ausgabe!

Die Insertion ist nicht vom Kauf des Heftes abhängig!

RESERVIERUNGS-SERVICE

Selbstverständlich denken wir bei diesem Assembler-Heft auch wieder an jene unter unseren Lesern, die keine Zeit haben, diese ganzen Listings einzugeben. Für sie hält der Kassettenservice das gesamte Angebot dieses Heftes auf drei Disketten bereit. Diese Disketten sind nur im Paket zu beziehen, sie kosten geschlossen 75,– DM. Einfach den Coupon auf der nächsten Seite ausschneiden, ausfüllen und absenden. Diese Seite ist so gestaltet, daß keine Information des Heftes verloren geht.

Ein Versand auf Kassette ist, wie Ihnen sicher bekannt, bei Assemblerprogrammen nicht möglich. Achtung: Volles Umtauschrecht bei Diskettenfehlern! Wir weisen ausdrücklich darauf hin, daß die Disketten nur nach Bestelleingang von Hand gefertigt werden und nicht in jedem Fall vorrätig sind. Deswegen kann es zu Lieferzeiten von bis zu zwei Wochen kommen.



ABO SERVICE-KARTE

TI

Ich nehme zur Kenntnis, daß die Belieferung erst beginnt, wenn die Abo-Gebühr dem Verlag zugegangen ist.

TI REVUE

Abo-Service
Postfach 1107
8044 UNTERSCHLEISSHEIM

Coupon

Ja, ich möchte von Ihrem Angebot Gebrauch machen.

Bitte senden Sie mir bis auf Widerruf ab sofort jeweils die nächsten zwölf Ausgaben an untenstehende Anschrift. Sollte ich nicht vier Wochen vor Ablauf schriftlich kündigen, läuft diese Abmachung automatisch weiter.

Name ____________________

Vorname ____________________

Straße/Hausnr. ____________________

Plz/Ort ____________________

Ich bezahle:

☐ per beiliegendem Verrechnungsscheck

☐ gegen Rechnung

☐ bargeldlos per Bankeinzug von meinem Konto

bei (Bank) und Ort ____________________

Kontonummer ____________________

Bankleitzahl ____________________

(steht auf jedem Kontoauszug)

Unterschrift ____________________

Von meinem Widerspruchsrecht habe ich Kenntnis genommen.

Unterschrift ____________________

PROGRAMMSERVICE

Hiermit bestelle ich in Kenntnis Ihrer Verkaufsbedingungen die Listings dieses Heftes auf

☐ 3 Disketten zum Preis von DM 75,–

Ich zahle:
per beigefügtem Scheck ( )
Gegen Bankabbuchung am Versandtag ( )
Zutreffendes bitte ankreuzen!

Meine Bank (mit Ortsname) ....

Meine Kontonummer ....

Meine Bankleitzahl .... (steht auf jedem Bankauszug)

Vorname .... Nachname ....

Str./Nr. .... PLZ / Ort ....

Hiermit bestätige ich mit meiner Unterschrift, Ihre Verkaufsbedingungen gelesen zu haben und zu akzeptieren.

Unterschrift ....

Bitte ausschneiden und einsenden an
TI-REVUE
Assembler-Service
Postfach 1107
8044 Unterschleißheim



REVUE

Das Magazin für TI 99-4A

Es geht! Dateien eröffnen und bearbeiten mit dem Kassetten-Recorder

30 Seiten Listings für Ihren TI 99/4A

Assembler leicht gemacht

Neue Software im Test

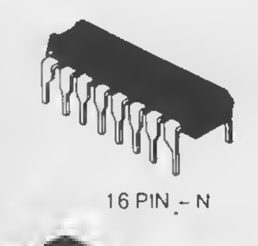
Drucker - richtig behandelt!

4 Seiten Anzeigen rund um den TI 99/4A

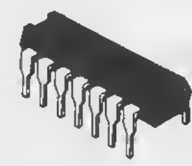


Gehäuse und thermische Angaben

PLASTIK



16 PIN – N



14 PIN – N



8 PIN – N



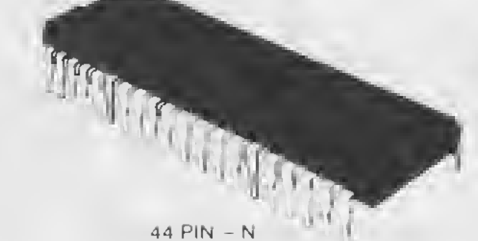
22 PIN – N



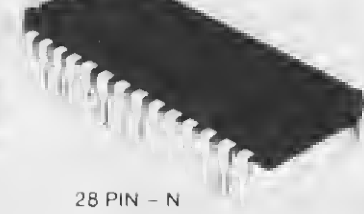
18 PIN – N



20 PIN – N



44 PIN – N



28 PIN – N

```
       MOVB R12,*R10      Byte in den Pattbuffer zur)ckschieben
*
PIXELA DECT @PRMZAL       Alle )bergebenen Werte berechnet ?
       JNE  PIXEL3        Wenn nicht, n(chstes Pixel
*
       CB   @BYT1,R7
       JNE  PIXELB
       MOV  @BACK1,R11
       RT
*
PIXELB LIMI 2             VDP-Interrupt wieder zulassen
       B    @BASIC        N(chste Basic Anweisung
*
* BL @BITMAP
*
* Videoprozessor auf BIT-MAP-MODE umschalten, den Pattbuffer in's VDP-
* -RAM bringen und Farbliste setzen.
*
BITMAP MOV  R11,@BACK1    Returnadresse sichern
*
       LI   R0,>2000      Farbliste auf gleiche V- und H-Farbe setzen,
       LI   R1,>2200      damit beim (ndern der VDP-Register nichts zu
       LI   R2,6144       sehen ist.
BITMP1 BLWP @VSBW         Byte in's VDP-RAM bringen
       INC  R0            Adresse plus eins
       DEC  R2            Alle Bytes gesetzt ?
       JNE  BITMP1        Wenn nicht, n(chstes Byte
*
       MOVB @VREG1+3,@VCOP1  VDP-Register 1 kopieren
       LI   R1,VREG1         Adresse der BIT-MAP-MODE Werte
       LI   R2,8          Acht Byte sind zu (ndern
BITMP2 MOV  *R1+,R0       Registernummer- und Wert nach R0
       BLWP @VWTR         Register (ndern
       DEC  R2            Alle Register ge(ndert ?
       JNE  BITMP2        Wenn nicht, n(chstes Register
*
* Bitmap-Namensliste setzen
*
       LI   R0,>1800      Adresse der Namensliste laden
       LI   R1,0          R1 = Zeichen 0
       LI   R2,3          Drei Drittel setzen
BITMP3 BLWP @VSBW         Zeichen )bertragen
       INC  R0            Adresse um eins erh!hen
       SWPB R1            Byte in Wort verwandeln
       INC  R1            Zeichen um eins erh!hen
       CI   R1,>0100      Ist R1 = 256 ?
       JEQ  BITMP4        Wenn ja, pr)fen ob noch ein Drittel frei ist
*
       SWPB R1            Wort in Byte verwandeln
       JMP  BITMP3        N(chstes Zeichen
BITMP4 LI   R1,0          Zeichencode wieder auf null zur)cksetzen
       DEC  R2            Alle Drittel gesetzt ?
       JNE  BITMP3        Wenn nicht, n(chstes Drittel
*
       CLR  R0            Grafic ab Adresse >0000 in's VDP-RAM bringen
       LI   R1,PATT       Adresse des P.D.T.-Buffers
       LI   R2,6144       6144 (>1800) Bytes sind zu )bertragen
       BLWP @VMBW         Schreiben
*
       LI   R0,>2000      Farbliste ab Adresse 8192 in's VDP-RAM bringen
       LI   R1,FARBE      Adresse des Farblisten-Buffers
       LI   R2,6144       6144 (>1800) Bytes sind zu )bertragen
       BLWP @VMBW         Schreiben
*
       MOV  @BACK1,R11    Returnadresse wiedergewinnen
       RT                 Return zum rufenden Programm
*
* BL @ENDE
*
* Diese Routine setzt die alten VDP-Register und die alten Werte in's
* VDP-RAM.
*
ENDE   MOVB @VREG2+3,@VCOP1  VDP-Register 1 kopieren
       LI   R1,VREG2         R1 mit Adresse der Registerwerte f)r G.-Mode laden
       LI   R2,8             Acht Register m)~en ge(ndert werden
*
ENDE1  MOV  *R1+,R0       Registernummer und Wert nach R0
       BLWP @VWTR         Register (ndern
       DEC  R2            Alle Register ge(ndert ?
       JNE  ENDE1         Wenn nicht, n(chstes Register
*
       LI   R0,1024       Standard-Zeichensatz wieder im VDP-RAM plazieren
       LI   R1,BUFFER     In R1 steht die Adresse des Buffer's
       LI   R2,768        Der Buffer ist 768 Bytes lang
       BLWP @VMBW         Bytes in's VDP-RAM schreiben
*
       MOVB @BYT0,@>83C2 Wirkung der QUIT-Taste wiederherstellen
       LIMI 2            VDP-Interrupt wieder zulassen
       LWPI GPLWS        GPL-Arbeitsregister laden
       BLWP @>0000       Zum Titelbildschirm zur)ckspringen
*
* BL @STRING
*
* Diese Routine )bernimmt Zeile und Spalte und setzt an diese
* Position im P.D.T.- oder Farb-Buffer einen Acht-Byte-String der aus
* den 16 Bytes gebildet wird, welche von Basic her )bergeben wurden.
*
STRING MOVB @LNKNUM,R12 Anzahl der Parameter nach R12
       LI   R1,1          1.Parameter
       BL   @TSTGR        Parameter )bernehmen und testen
       DATA 1,24
       MOV  @FAC,R13      Zeile
*
       INC  R1            2.Parameter
       BL   @TSTGR        Parameter )bernehmen und testen
       DATA 1,32
       MOV  @FAC,R14      Spalte
*
       CLR  R0            R0=0
       INC  R1            3.Parameter
       LI   R2,SATZ       R2 mit Stringbuffer-Adresse laden
       MOVB @BYT16,*R2    Maximal 16 Bytes )bernehmen
       BLWP @STRREF       String )bernehmen
       CB   *R2,@BYT16    Sind 16 Bytes )bergeben ?
       JNE  ERRT2         Wenn nicht, Fehlermeldung
*
       LI   R15,1         Nur 1 Durchgang wenn kein 4.Parameter vorhanden
       SRL  R12,8         Byte in Wort verwandeln
```

```
        CI   R12,4         Sind 4 Parameter vorhanden ?
        JNE  STRIG1        Wenn nicht, sofort mit dem Programm beginnen
*
        INC  R1            4.Parameter
        BL   @TSTGR        Parameter }bernehmen und testen
        DATA 1,76B
        MOV  @FAC,R15      Parameter nach R15, gleichzeitig Schleifenz{hler
        JMP  STRIG1        Mit dem eigentlichen Programm beginnen
*
ERRT2   LIMI 2             VDP-Interrupt wieder zulassen
        B    @BARERR       Fehlermeldung ausgeben und Abbruch
*
STRIG1  LI   R10,16        R10=16
        LI   R9,SATZ+1     R9 mit Adresse Stringbuffer's +1 laden
*
STRIG2  CLR  R5            R5=0
        MOVB *R9,R5        Ein Byte aus Stringbuffer nach R5
        SWPB R5            Byte in Wort verwandeln
        CI   R5,48         Ist R5 mind. 4B ?
        JLT  ERRT2         Wenn nicht, Fehlermeldung
        CI   R5,70         Ist R5 max. 70 ?
        JGT  ERRT2         Wenn nicht, Fehlermeldung
        CI   R5,64         Wenn R5 gr|~er 64, dann sind es Buchstaben
        JGT  STRIG3
        CI   R5,58         Wenn R5 kleiner 5B, dann sind es Zahlen
        JLT  STRIG4
        JMP  ERRT2         Wenn alles nicht zutrifft, Fehlermeldung
*
STRIG3  SWPB R5            Wort in Byte verwandeln
        AB   @BYTM55,R5    Buchstabenkonstante addieren
        MOVB R5,*R9+       Byte zur}ck in Stringbuffer und Adresse +1
        JMP  STRIG5
*
STRIG4  SWPB R5            Wort in Byte verwandeln
        AB   @BYTM48,R5    Zahlenkonstante addieren
        MOVB R5,*R9+       Byte zur}ck in Stringbuffer und Adresse +1
*
STRIG5  DEC  R10           Alle Bytes bearbeitet ?
        JNE  STRIG2        Wenn nicht, n{chstes Byte
*
        MOV  @BUFADR,R5    R5 mit aktueller Buffer-Adresse laden
        MOV  R13,R0        Zeilenwert nach R0 kopieren
        DEC  R0            R0=R0-1
        SLA  R0,8          R0=R0*256
        MOV  R14,R1        Spaltenwert nach R1 kopieren
        DEC  R1            R1=R1-1
        SLA  R1,3          R1=R1*8
        A    R1,R0         Beide Werte addieren ; Ergebnis ist in R0
        A    R5,R0         P.D.T.-Buffer-Adresse dazu
        LI   R4,>1800      R4=6144
        A    R4,R5         Grenzwert in R5 ablegen
*
        LI   R9,MATRIX     R9 mit Variablenadresse laden
        LI   R10,8         Acht Bytes m}~en gebildet werden
        LI   R11,SATZ+1    R11 mit Stringbufferadresse +1 laden
*
STRIG6  CLR  R12           R12=0
        MOVB *R11+,R12     Ein Byte nach R12
        MOVB *R11+,R13     Zweites Byte nach R13
        SLA  R12,4         Wert in R12 an sign. Stelle schieben
        SOCB R13,R12       Beide Werte durch log. ODER verkn}pfen
        MOVB R12,*R9+      Fertiges Byte in Variablenbuffer packen
        DEC  R10           Sind acht Bytes verarbeitet ?
        JNE  STRIG6        Wenn nicht, n{chstes Byte
*
STRIG7  LI   R10,8         R10=B
        LI   R9,MATRIX     R9 mit Variablenadresse laden
*
STRIG8  MOVB *R9+,*R0+     Byte aus Variable in den P.D.T.-Buffer bringen
        DEC  R10           Ein vollst{ndiges Zeichen }bertragen ?
        JNE  STRIG8        Wenn nicht, n{chstes Byte
*
        C    R0,R5         Ende des P.D.T.-Buffer's erreicht ?
        JEQ  STRIG9        Wenn ja, n{chste Basic-Anweisung
*
        DEC  R15           Sind alle Wiederholungen ausgef}hrt ?
        JNE  STRIG7        Wenn nicht, n{chste Runde
*
STRIG9  LIMI 2             VDP-Interrupt zulassen
        B    @BASIC        N{chste Basic-Anweisung
*
* BL @SQUARE
*
* Dieses Unterprogramm ersetzt die im GPL-Interpreter ent-
* haltene, aber sehr langsame Wurzel-Routine.
*
*
SQUARE  MOV  @FAC,R2       Wert nach R2 }bertragen
        C    @D16257,R2    Ist der Wert kleiner als 127^2 ?
        JGT  SQR4          Wenn ja, dann 0 bis 127 ausprobieren
*
        LI   R0,SQRDA2     R0 mit Werten f}r 12B bis 256 laden
SQR1    C    *R0+,R2       Ist der Wert aus der Tabelle kleiner ?
        JLT  SQR2          Wenn ja, dann n{chsten Wert
        JMP  SQR3          Wenn nicht, dann ist die richtige Wurzel gefunden
*
SQR2    INCT R0            Sprung zum n{chsten Oberwert
        JMP  SQR1          Wert untersuchen
*
SQR3    MOV  *R0,@FAC      Richtigen Wert nach FAC
        JMP  SQR8          Zur}ck zum rufenden Programm
*
SQR4    LI   R0,SQRDA1     R0 mit Werten f}r 0 bis 127 laden
SQR5    C    *R0+,R2       Ist der Wert aus der Tabelle kleiner ?
        JLT  SQR6          Wenn ja, dann n{chsten Wert
        JMP  SQR7          Wenn nicht, dann ist die richtige Wurzel gefunden
*
SQR6    INCT R0            Sprung zum n{chsten Oberwert
        JMP  SQR5          Wert untersuchen
*
SQR7    MOV  *R0,@FAC      Richtigen Wert nach FAC
*
SQRB    RT                 Zur}ck zum rufenden Programm
*
* Ende Segment "BITMAP1S"
*
```

```
* Segment "@BITMAP2S"
*
* *****************
* *               *
* * Unterprogramme *
* *               *
* *     2.Teil    *
* *               *
* *****************
*
* BL @SHUT
*
* Dieses Unterprogramm schlie~t eine Datei
*
SHUT   MOVB @BYT1,R1     Datei-Schlie~-Befehl nach R1
       MOV  @PAB,RØ      In RØ steht die Adresse des Pab im VDP-RAM
       BLWP @VSBW        Byte in's VDP-RAM bringen
       BL   @INOUT       Datei schlie~en
       B    @BASIC       N{chste Basic-Anweisung
*
* BL @GETSTR
*
* }bernimmt den Device-Namen von Basic
*
GETSTR MOV  R11,@BACK1   Returnadresse sichern
       LI   R1,1         Parameternummer in R1 laden
       CLR  RØ           RØ=Ø
       LI   R2,SATZ      R2 mit Adresse des Stringbuffers laden
       MOVB @BYT15,*R2   Maximal 15 Bytes sollen }bernommen werden
       BLWP @STRREF      Device-Namen }bernehmen
       MOV  @PAB,RØ      In PAB+9 mu~ die L{nge des Dateinamen stehen
       A    @DØ9,RØ
       LI   R1,SATZ      In R1 steht die Adresse des Stringbuffers
       MOVB *R1,R2       Aktuelle Stringl{nge nach R2
       CB   @BYTØ,R2     Ist der String null Bytes lang
       JEQ  GETSR1       Wenn ja, Programmabbruch
*
       SRL  R2,B         Byte in Wort verwandeln
       INC  R2           Stringl{nge mu~ mit}bertragen werden
       BLWP @VMBW        Stringl{nge und String an den Pab anh{ngen
       MOV  @BACK1,R11   Returnadresse wiedergewinnen
       RT                Zur}ck zum rufenden Programm
*
GETSR1 B    @BASIC       Eintrittspunkt f}r Programmabbruch bei Nullstring
*
* BL @RESERV
*
* Platz f}r den Pab und den Pab-Buffer im VDP-RAM reservieren
*
RESERV MOVB @BYTØ,@GPLSTA  GPL-Statusbyte l|schen
       LWPI GPLWS          GPL-Arbeitsregister laden
       BLWP @GPLLNK        Platz im VDP-RAM reservieren
       DATA RES
*
       LWPI MYREGI         Eigene Arbeitsregister laden
       MOV  @>B31A,@PAB    Die erste freie Adresse in die Variable
       MOV  @PAB,@PABBUF
       A    R5,@PABBUF
       MOV  @PAB,RØ        Adresse des Pab nach RØ
       MOV  RØ,@POINT      Pab-Pointer setzen
       MOV  @BUFADR,R1     Dateidaten-Adresse nach R1
       BLWP @VMBW          Bytes in's VDP-RAM bringen
       INCT RØ             Ab PAB+2 steht die Adresse des Pab-Buffers
       LI   R1,PABBUF      R1 mit der Adresse der Variablen laden
       LI   R2,2           Zwei Bytes sind zu }bertragen
       BLWP @VMBW          Bytes in's VDP-RAM bringen
       RT                  Zur}ck zum rufenden Programm
*
* BL @BRING
*
* Dieses Unterprogramm bringt den neuen Seikosha-String
* in den PAB-Buffer.
*
BRING  MOV  @PABBUF,RØ     RØ mit der Adresse des Pabbuffer's laden
       MOV  R15,R12        Wert von R15 nach R12 kopieren
       SLA  R12,3          R12=Wert von R12 mal 8
       A    R12,RØ         R12=R12+Adresse von PABBUF
       LI   R1,STRINØ      R1 mit Adresse von STRINØ laden
       LI   R2,8           Acht Byte sind zu }bertragen
       BLWP @VMBW          Byte's in das VDP-RAM schreiben
       MOV  @BACK1,R11     Returnadresse wiedergewinnen
       RT                  Zur}ck zum rufenden Programm
*
* BL @DRUCK1
*
* Wenn die erste H{lfte einer Bildschirmzeile f}r den Drucker
* aufbereitet wurde, wird sie durch dieses Unterprogramm
* ausgedruckt und der Drucker f}r die n{chste H{lfte vor-
* bereitet.
*
DRUCK1 MOV  R11,@BACK1     Returnadresse sichern
       LI   R1,>BØØØ       12B Bytes sollen zum Drucker geschickt werden
       MOV  @PAB5,RØ
       BLWP @VSBW
       BL   @INOUT         Daten zum Drucker schicken
       LI   R1,>Ø4ØØ       Vier Bytes sollen zum Drucker geschickt werden
       MOV  @PAB5,RØ
       BLWP @VSBW
       MOV  @PABBUF,RØ     Daten f}r Druckposition in den PAB-Buffer laden
       LI   R1,MYDAT2
       LI   R2,4           Vier Bytes sind zu }bertragen
       BLWP @VMBW          Bytes in's VDP-RAM schreiben
       BL   @INOUT         Daten zum Drucker schicken
       MOV  @BACK1,R11     Returnadresse wiedergewinnen
       RT                  Zur}ck zum rufenden Programmm
*
* BL @DRUCK2
*
* Wenn die zweite H{lfte einer Bildschirmzeile f}r den Drucker
* aufbereitet wurde, wird sie durch dieses Unterprogramm
* ausgedruckt und der Drucker f}r die n{chste Zeile vor-
* bereitet.
*
DRUCK2 MOV  R11,@BACK1     Returnadresse sichern
       LI   R1,>8ØØØ       128 Bytes sollen zum Drucker geschickt werden
       MOV  @PAB5,RØ
       BLWP @VSBW
       BL   @INOUT         Daten zum Drucker schicken
```

```
       LI   R1,>Ø5ØØ     F}nf Bytes sollen zum Drucker geschickt werden
       MOV  @PAB5,RØ
       BLWP @VSBW
       MOV  @PABBUF,RØ   Daten f}r Druckposition in den PAB-Buffer laden
       LI   R1,MYDAT1
       LI   R2,5         F}nf Bytes sind zu }bertragen
       BLWP @VMBW        Bytes in's VDP-RAM schreiben
       BL   @INOUT       Daten zum Drucker schicken
       MOV  @BACK1,R11   Returnadresse wiedergewinnen
       RT                Zur}ck zum rufenden Programm
*
* BL @DSGNCH
*
* Mit Hilfe dieses Unterprogramms wird aus der Bitmatrix der neue
* Seikosha-String berechnet.
*
DSGNCH MOV  R11,@BACK1   Returnadresse sichern
       CLR  RØ           RØ=Ø
       LI   R1,STRINØ    R1 mit der Adresse von STRINØ laden
       LI   R2,4         4 mal soll die folgende Schleife durchlaufen werden
DSGN1  MOV  RØ,*R1+      Ein Wort l|schen
       DEC  R2           Schleife beendet ?
       JNE  DSGN1        Wenn noch nicht alles gel|scht ist, n{chstes Wort
*
       LI   R9,B         Schleifenz{hler f}r ein Byte
       LI   R12,STRINØ   R12=Adresse von STRINØ
       LI   R15,MATRIX   R15=Adresse der Bitmatrix
       LI   R4,B         Z{hlkonstante f}r aktuelle Bitmatrixadresse
DSGN2  MOV  R15,R1Ø      Bitmatrixadresse nach R1Ø kopieren
       LI   R11,6        Z{hler f}r zu berechnende Bits
       LI   RØ,1         RØ mit Verschiebewert laden
       CLR  R8           R8=Ø
       MOVB *R1Ø,RB      Bit Ø von der Bitmatrix nach RB kopieren
       A    R4,R1Ø       Aktuelle Bitmatrixadresse um acht erh|hen
DSGN3  MOVB *R1Ø,R13     N{chstes Bit nach R13 bringen
       SWPB R13          Byte in Wort verwandeln
       SLA  R13,Ø        R13 wird nun um Wert von RØ nach links verschoben
*                        --> R13 mal 2^Wert von RØ
       SWPB R13          Wort in Byte verwandeln
       AB   R13,RB       Die einzelnen Bits zu einem Byte aufaddieren
       A    R4,R1Ø       Aktuelle Bitmatrixadresse um acht erh|hen
       INC  RØ           Verschiebewert um eins erh|hen
       DEC  R11          Sind sieben Bits berechnet ?
       JNE  DSGN3        Wenn nicht, n{chstes Bit
*
       AB   @BYT12B,RB   Da das achte Bit bei dem Seikosha GP 1ØØ A immer
*                        auf 1 sein mu~, werden hier 128 zu dem Byte
*                        hinzuaddiert.
       MOVB RB,*R12      Fertiges Byte in den String-Buffer packen
       INC  R15          Urspr}ngliche Bitmatrixadresse um eins erh|hen
       INC  R12          String-Bufferadresse um eins erh|hen
       DEC  R9           Sind acht Bytes berechnet ?
       JNE  DSGN2        Nein, also n{chstes Byte berechnen
*
       LWPI MYREGI       Normale Arbeitsregister wieder laden
       BL   @BRING       Seikosha-String in den Pab-Buffer bringen
*
* BL @COPIER
*
* Dieses Unterprogramm zerlegt ein Byte in Bits.
*
COPIER SRL  R5,8         Byte in Wort verwandeln
       SRL  R5,Ø         R5 um Wert von RØ verschieben (dividieren)
       SWPB R5           Wort in Byte verwandeln
       MOVB R5,*R15+     Bit in die Bitmatrix schreiben un Adr. + 1
       SWPB R5           Byte in Wort verwandeln
       SLA  R5,Ø         R5 um Wert von RØ verschieben (multiplizieren)
       SWPB R5           Wort in Byte verwandeln
       SB   R5,R7        Aktuellen Restwert gewinnen
       MOVB R7,R5        Restwert nach R5 kopieren
       DEC  RØ           Ist das Byte zerlegt ?
       JNE  COPIER       Wenn nicht, n{chstes Bit
*
       MOVB R5,*R15+     Bit 7 kann nur noch 1 oder Ø sein
       RT                Zur}ck zum rufenden Programm
*
* BL @INOUT
*
* Dieses Unterprogramm f}hrt die Datei-Operation durch
*
INOUT  MOV  R11,@BACK2   Returnadresse sichern
*
       LWPI MYREG2       2.Satz Arbeitsregister laden
       MOV  @PAB,R6      R6 enth{lt die Adresse der Datei Namenl{nge
       A    @DØ9,R6
       MOV  R6,@PNTR     L{ngenzeiger setzen
       MOVB @BYTØ,@GPLSTA  GPL-Statusbyte l|schen
       BLWP @DSRLNK      File-Operation durchf}hren
       DATA B
       B    @STAND       Operation auf I/O-Fehler untersuchen
*
* BL @STAND
*
* Statusbits auf Dateifehler }berpr}fen
*
STAND  MOV  @PAB,RØ      In PAB+1 sind die Statusbits
       INC  RØ
       BLWP @VSBR        Also Byte holen
       SRL  R1,13        Bit Ø bis Bit 2 m}ssen isoliert werden
       SWPB R1           Wort in Byte verwandeln
       CB   @BYTØ,R1     Sind die Statusbits null ?
       JNE  STAND1       Wenn nicht, Fehlercode festlegen
*
       MOVB @GPLSTA,RØ   GPL-Statusbyte kopieren
       SLA  RØ,2         Bit 2 (COND-BIT) isolieren
       SRL  RØ,15        Bit an Nicht-Sign. Stelle im Nicht-Sign.Byte
       SWPB RØ           Wort in Byte verwandeln
       CB   @BYTØ,RØ     Ist das COND-BIT null ?
       JNE  STAND1       Wenn nicht, I/O-Fehler Ø ausgeben
*
       LWPI MYREGI       Wieder die alten Arbeitsregister laden
       MOV  @BACK2,R11   Returnadresse wiedergewinnen
       RT                Zur}ck zum rufenden Programm
*
STAND1 MOV  @PAB,R2      R2 mit der Adresse des Pab laden
       MOV  R2,@POINT    Adresse in den Pab-Pointer schieben
       MOV  R1,RØ        Statusbit ist gleichzeitig Fehlercode
       BLWP @ERR         Fehler ausgeben und Abbruch
```

```
*
* BL @TRANS
*
* Dieses Unterprogramm kann acht Byte von einer Adresse zu einer anderen
* Adresse transferieren.
*
TRANS  MOV  *R11+,R6        Startadresse nach R6
       MOV  *R11+,R7        Zieladresse nach R7
       LI   RØ,4            Vier Worte sind zu transferieren
TRANS1 MOV  *R6+,*R7+       Wort transferieren
       DEC  RØ              Sind alle vier Worte }bertragen
       JNE  TRANS1          Wenn nicht, n{chstes Wort
*
       RT                   Zur}ck zum rufenden Programmteil
*
* BL @SUBLI1
*
* Dieses Unterprogramm beinhaltet Routinen f}r das Hauptprogramm 'LINE'
*
SUBLI1 MOV  R11,@BACK2      Returnadresse sichern
       BL   @TRANS          Acht-Byte-Block transferieren
       DATA XAFP,FAC
       BLWP @XMLLNK         Gleitkomma-Addition durchf}hren
       DATA FADD
       BL   @TRANS          Acht-Byte-Block transferieren
       DATA FAC,XAFP
       BLWP @XMLLNK         Gleitkomma in Integer verwandeln
       DATA CFI
       MOV  @FAC,@SATZ+2    X-Koordinate nach Satz plus zwei
       MOV  @YA,@SATZ       Y-Koordinate nach Satz
       MOV  @DØ2,@PRMZAL    Zwei Koordinaten sollen bearbeitet werden
       LWPI MYREG2          Zweiten Satz Arbeitsregister laden
       MOVB @BYT1,R7        Die Koordinaten kommen von au~erhalb
       BL   @SETTO          Koordinaten setzen
*
       LWPI MYREGI          Normale Arbeitsregister wieder laden
       MOV  @BACK2,R11      Returnadresse wiedergewinnen
       RT                   Zur}ck zum rufenden Programm
*
* BL @SUBLI2
*
* Dieses Unterprogramm beinhaltet Routinen f}r das Hauptprogramm 'LINE'.
*
SUBLI2 MOV  R11,@BACK2      Returnadresse sichern
       BL   @TRANS          Acht-Byte-Block transferieren
       DATA YAFP,FAC
       BLWP @XMLLNK         Gleitkomma-Addition durchf}hren
       DATA FADD
       BL   @TRANS          Acht-Byte-Block transferieren
       DATA FAC,YAFP
       BLWP @XMLLNK         Gleitkomma in Integer verwandeln
       DATA CFI
       MOV  @FAC,@SATZ      Y-Koordinate nach Satz
       MOV  @XA,@SATZ+2     X-Koordinate nach Satz plus zwei
       MOV  @DØ2,@PRMZAL    Zwei Koordinaten sollen bearbeitet werden
       LWPI MYREG2          Zweiten Satz Arbeitsregister laden
       MOVB @BYT1,R7        Die Koordinaten kommen von au~erhalb
       BL   @SETTO          Koordinaten setzen
*
       LWPI MYREGI          Normale Arbeitsregister wieder laden
       MOV  @BACK2,R11      Returnadresse wiedergewinnen
       RT                   Zur}ck zum rufenden Programm
*
* BL @GTPRM
*
* Numerische Parameter holen
*
GTPRM  BLWP @NUMREF         Parameter holen
       BLWP @XMLLNK         Gleitkomma in Integer verwandeln
       DATA CFI
       RT                   Return zum rufenden Programm
*
* BL @TSTGR
*
* ]berpr}ft die }bergebenen Werte auf Zul{~igkeit
*
TSTGR  MOV  *R11+,R2        Untere Grenze
       MOV  *R11+,R3        Obere Grenze
       MOV  R11,@BACK1      Returnadresse sichern
       DEC  R2              Untere Grenze minus eins
       INC  R3              Obere Grenze plus eins
       CLR  RØ              RØ=Ø .
       BL   @GTPRM          Parameter }bernehmen
       MOV  @FAC,R4         Parameter nach R4 kopieren
       C    R4,R2           Ist der Wert gr|~er oder gleich UG ?
       JGT  TSTGR1          Wenn ja, obere Grenze testen
*
       B    @BARERR         Wenn nicht, Fehlermeldung
TSTGR1 C    R4,R3           Ist der Wert kleiner oder gleich OG ?
       JLT  TSTGR2          Wenn ja, zur}ck zum rufenden Programm
*
       B    @BARERR         Wenn nicht, Fehlermeldung ausgeben
TSTGR2 MOV  @BACK1,R11      Returnadresse wiedergewinnen
       RT                   Zur}ck zum rufenden Programm
*
* B @BARERR
*
* Unterprogramm Fehlermeldung ausgeben.
*
BARERR LI   RØ,ERR22        RØ mit Fehlercode f}r 'BAD ARGUMENT' laden
       BLWP @ERR
*
* B @BASIC
*
* Dieses Unterprogramm springt in's Basic zur}ck.
*
BASIC  MOVB @BYTØ,@GPLSTA   GPL-Statusbyte l|schen
       LWPI GPLWS           GPL-Arbeitsregister laden
       B    @>ØØ7Ø          N{chste Basic-Anweisung
*
* Ende Segment,"BITMAP2S"
*
```

```
* Segment "@BITMAP3S"
*
* *****************
* *               *
* * Hauptprogramme *
* *               *
* *     1.Teil    *
* *               *
* *****************
*
* CALL LINK("CLEAR")
*
* Dieses Programm l|scht den Pat-Buffer und sichert die Standard-Zeichen.
*
CLEAR  LIMI Ø             VDP-Interrupt unterdr}cken
*
       LWPI MYREGI        Eigene Arbeitsregister laden
       LI   RØ,FARBE      RØ mit Adresse des Farbbuffer's laden
       LI   R1,>1212      Vordergrund : schwarz ; Hintergrund : mittelgr}n
       LI   R2,3Ø72       3Ø72 Worte schreiben
CLEAR1 MOV  R1,*RØ+       Wort in den Farbbuffer schreiben und Adresse +2
       DEC  R2            Alle 3Ø72 Worte geschrieben ?
       JNE  CLEAR1        Wenn nicht, n{chstes Wort
*
       LI   RØ,PATT       RØ mit Adresse des P.D.T.-Buffer's laden
       CLR  R1            R1=Ø
       LI   R2,3Ø72       3Ø72 Worte sind zu l|schen
CLEAR2 MOV  R1,*RØ+       Wort in den P.D.T.-Buffer schreiben und Adr. +2
       DEC  R2            Alle 3Ø72 Worte geschrieben ?
       JNE  CLEAR2        Wenn nicht, n{chstes Wort
*
       LI   RØ,1Ø24       Ab 1Ø24 steht der Standard-Zeichensatz
       LI   R1,BUFFER     R1 mit Adresse des Zeichensatz-Buffer's laden
       LI   R2,768        768 Bytes sind zu l|schen
       BLWP @VMBR         Bytes aus VDP-RAM lesen
       LIMI 2             VDP-Interrupt wieder zulassen
       B    @BASIC        N{chste Basic-Anweisung
*
* CALL LINK("SETPIX",Y-Koordinate,X-Koordinate,(bis zu 7 mal))
*
* Dieses Programm setzt Pixel.Dazu wird in 'P' eine 1 abgelegt.
*
SETPIX LIMI Ø             VDP-Interrupt unterdr}cken
*
       LWPI MYREGI        Eigene Arbeitsregister laden
       MOVB @LNKNUM,@ANZAHL  Anzahl der Parameter sichern
       MOVB @BYT1,@P         In 'P' steht eine 1 f}r Pixel setzen
       B    @PIXEL           Pixel setzen
*
* CALL LINK("CLRPIX",Y-Koordinate,X-Koordinate,(bis zu 7 mal))
*
* Dieses Programm l|scht Pixel.Dazu wird in 'P' eine Ø abgelegt.
*
CLRPIX LIMI Ø             VDP-Interrupt unterdr}cken
*
       LWPI MYREGI        Eigene Arbeitsregister laden
       MOVB @LNKNUM,@ANZAHL  Anzahl der Parameter sichern
       MOVB @BYTØ,@P         In 'P' steht eine Ø f}r Pixel l|schen
       B    @PIXEL           Pixel l|schen
*
* CALL LINK("ZEIG",Sekunden)
*
* Nach Aufruf dieses Programm's wird die im Buffer erstellte Grafic
* f}r eine bestimmte Zeit in's VDP-RAM gebracht.
*
ZEIG   LIMI Ø             VDP-Interrupt unterdr}cken
*
       LWPI MYREGI        Eigene Arbeitsregister laden
       MOVB @BYT16,@>83C2 Wirkung der QUIT-Taste aufheben
       LI   R1,1          1.Parameter
       BL   @TSTGR        Parameter holen und testen
       DATA 1,48Ø         UG,OG
       MOV  @FAC,R12      Parameter nach R12 kopieren
       BL   @BITMAP       Videoprozessor auf Bitmap umschalten
       CLR  RØ            RØ=Ø
       MOVB @BYTØ,@WATCH  VDP-Interrupt-Timer zur}cksetzen
       LIMI 2             VDP-Interrupt-Timer starten
ZEIG1  CB   @WATCH,@BYT5Ø Sind 5Ø Impulse f}r eine Sekunde abgelaufen ?
       JNE  ZEIG1         Wenn nicht, n{chste Abfrage
*
       MOVB @BYTØ,@WATCH  Wenn doch, VDP-Interr.-Timer zur}cksetzen
       INC  RØ            RØ=RØ+1
       C    RØ,R12        Ist die Zeit vorbei
       JNE  ZEIG1         Wenn nicht, n{chster Durchgang
*
       LIMI Ø             VDP-Interrupt unterdr}cken
       B    @ENDE         Videoprozessor wieder auf Grafic-Modus umschalten
*
* CALL LINK("WRITE",Ø oder 1,Zeile,Spalte,"String")
*
* Dieses Programm l{~t ein schreiben in die Grafik zu.
* Erlaubt sind die Zeichen 32 bis 127.
* Wenn erste Parameter 1, dann Text hinein kopieren
* Wenn erste Parameter Ø, dann Hintergrund vorher l|schen
*
WRITE  LIMI Ø             VDP-Interrupt unterdr}cken
*
       LWPI MYREGI        Eigene Arbeitsregister laden
       LI   R1,1          1.Parameter
       BL   @TSTGR        Parameter }bernehmen und testen
       DATA Ø,1
       MOV  @FAC,R14      Parameter in R14 ablegen
       SWPB R14           Wort in Byte verwandeln
       INC  R1            2.Parameter (Zeile)
       BL   @TSTGR        Parameter }bernehmen und testen
       DATA 1,24
       MOV  @FAC,R9       Parameter nach R9
       INC  R1            3.Parameter
       BL   @TSTGR        Parameter }bernehmen und testen
       DATA 1,32
       MOV  @FAC,R1Ø      Parameter nach R1Ø
       CLR  RØ            RØ=Ø
       INC  R1            4.Parameter (String)
       LI   R2,SATZ       R2 mit Adresse des Stringbuffer's laden
       MOVB @BYT32,*R2    Maximal 32 Zeichen }bernehmen
       CLR  R12           R12=Ø
       BLWP @STRREF       String }bernehmen
       LI   R2,SATZ       R2 mit Adresse des Stringbuffer's laden
```

```
       MOVB *R2+,R12     Aktuelle Stringl{nge nach R12 (Schleifenz{hler)
       SWPB R12          Byte in Wort verwandeln
       CI   R12,Ø        Leerstring ?
       JNE  WRITE1       Wenn nicht, dann schreiben
*
       B    @BASIC       Wenn Leerstring, dann zur}ck in's Basic
WRITE1 LI   R5,BUFFER    Adresse des Pattern-Buffer in R5 laden
       MOV  R9,RØ        Zeile nach RØ
       DEC  RØ           RØ=RØ-1    (Zeile - 1)
       SLA  RØ,8         RØ=RØ*256  (Zeilenwert * 256)
       MOV  R1Ø,R1       Spalte nach R1
       DEC  R1           R1=R1-1    (Spalte - 1)
       SLA  R1,3         R1=R1*8    (Spaltenwert * 8)
       A    R1,RØ        RØ=RØ+R1   (Anzeigeadresse  in RØ ablegen)
       LI   R3,PATT      Patt-Buffer-Adresse in R3 laden
       A    R3,RØ        RØ=RØ+R3   (Buffer-Basisadresse hinzu)
WRITE2 CLR  R4           R4=Ø
       MOVB *R2+,R4      Byte aus String-Variablen nach R4 und Adresse +1
       SB   @BYT32,R4    R4=R4-32   (Zeichen Ø bis 31 nicht darstellbar)
       SWPB R4           Byte in Wort verwandeln
       CI   R4,Ø         Ist Zeichen jetzt mind. null ?
       JLT  ERRT3        Wenn kleiner, dann Fehlermeldung
*
       CI   R4,95        Ist Zeichen jetzt max. 95 ?
       JGT  ERRT3        Wenn gr|~er, dann Fehlermeldung
*
       SLA  R4,3         R4=R4*8    (Zeichenwert * 8)
       A    R5,R4        R4=R4+R5   (Zeichenbuffer-Adresse dazu)
       LI   R1,8         Ein Zeichenstring ist acht Byte lang
WRITE3 CLR  R6           R6=Ø
       CLR  R7           R7=Ø
       MOVB *R4+,R6      Byte aus Zeichenbuffer nach RØ und Adr. +1
       MOVB *RØ,R7       Byte aus Pattbuffer nach R7
       CB   @BYTØ,R14    Soll der Hintergrund gel|scht werden ?
       JEQ  WRITE4       Ja, also Wert nicht verkn}pfen
*
       SOCB R6,R7        Sign. Bytes durch log. ODER verkn}pfen
       JMP  WRITE5       Byte in Pattbuffer bringen
*
WRITE4 MOVB R6,*RØ+      Hintergrund l|schen, also Byte sofort in Pattbuffer
       JMP  WRITE6       ]berpr}fe, ob String vollst{ndig
*
WRITE5 MOVB R7,*RØ+      Log. verkn}pften Wert in Pattbuffer bringen
WRITE6 DEC  R1           Ist der Zeichenstring vollst{ndig ?
       JNE  WRITE3       Wenn nicht, n{chstes Byte
*
       DEC  R12          Gesamter Anzeigestring auf dem Bildschirm ?
       JNE  WRITE2       Wenn nicht, n{chstes Zeichen
*
       LIMI 2            VDP-Interrupt zulassen
       B    @BASIC       N{chste Basic-Anweisung
ERRT3  LIMI 2            VDP-interrupt zulassen
       B    @BARERR      Fehlermeldung und Abbruch
*
* CALL LINK("CHAR",Zeile,Spalte,"String",Wiederholungen)
*
* Dieses Programm erlaubt es, ein Zeichen durch einen String
* umzudefinieren (genau wie CALL CHAR in Basic).Es sind maximal 768
* Wiederholungen erlaubt.
```

```
*
CHAR   LIMI Ø            VDP-Interrupt unterdr}cken
*
       LWPI MYREGI       Eigene Arbeitsregister laden
       LI   R1,PATT      R1 mit P.D.T.-Buffer-Adresse laden
       MOV  R1,@BUFADR   Adresse sichern
       BL   @STRING      In's eigentliche Programm springen
*
* CALL LINK("COLOR",Vordergrundfarbe,Hintergrundfarbe)
*                ,Zeile,Spalte,"String",Wiederholungen)
*
* Dieses Programm erlaubt es die Vordergrund- und Hintergrundfarbe
* des ganzen Bildschirm's auf einmal zu {ndern oder aber jede Zeichen-
* position farblich mittels eines Strings zu definieren.
*
COLOR  LIMI Ø            VDP-Interrupt unterdr}cken
*
       LWPI MYREGI       Eigene Arbeitsregister laden
       MOVB @LNKNUM,RØ   Anzahl der Parameter sichern
       CB   @BYT2,RØ     Sind zwei Parameter }bergeben worden ?
       JEQ  COLOR1       Wenn ja, dann Farbbuffer {ndern
*
       LI   R1,FARBE     R1 mit Adresse des Farbbuffer's laden
       MOV  R1,@BUFADR   Adresse zwischenspeichern
       BL   @STRING      String in den Farbbuffer bringen
COLOR1 LI   R1,1         1.Parameter
       BL   @TSTGR       Parameter }bernehmen und testen
       DATA 1,16
       MOV  @FAC,R13     Vordergrundfarbe nach R13
       INC  R1           2.Parameter
       BL   @TSTGR       Parameter }bernehmen und testen
       DATA 1,16
       MOV  @FAC,R14     Hintergrundfarbe nach R14
       DEC  R13          V-Farbe -1
       DEC  R14          H-Farbe -1
       SWPB R13          Wort in Byte verwandeln
       SWPB R14          Wort in Byte verwandeln
       SLA  R13,4        V-Farbe um vier Bits nach links schieben
       SOCB R14,R13      Beide Werte mit log. ODER verkn}pfen
       CLR  R1           R1=Ø
       MOVB R13,R1       Fertiges Byte nach R1
       LI   RØ,FARBE     RØ mit der Adresse des Farbbuffer's laden
       LI   R2,6144      6144 Bytes sind zu {ndern
COLOR2 MOVB R1,*RØ+      Byte in den Farbbuffer und Adresse +1
       DEC  R2           Sind alle Bytes ge{ndert ?
       JNE  COLOR2       Wenn nicht, n{chstes Byte
*
       LI   R13,1        R13=1
       SWPB R13          Wort in Byte verwandeln
       SLA  R13,4        R13=R13*16
       SOCB R14,R13      Beide Werte mit log. ODER verkn}pfen
       LI   RØ,VREG1+15  Zieladresse laden
       MOVB R13,*RØ      Berechneten Wert im Ziel speichern
       LIMI 2            VDP-Interrupt wieder zulassen
       B    @BASIC
*
* CALL LINK("HCOPY")
*
```

```
* Dieses Programm erlaubt es, eine Bildschirm-Hardcopy auf einem
* Seikosha GP 100 A auszugeben.
*
HCOPY   LWPI MYREGI        Eigene Arbeitsregister laden
        LI   R15,PDATA2     Adresse der aktuellen PAB-Daten laden
        MOV  R15,@BUFADR    Adresse sichern
        MOV  @D152,@>830C   152 Bytes sollen im VDP-RAM reserviert werden
        LI   R2,16          Die Pabdaten sind sechzehn Bytes lang
        MOV  @D16,R5
        BL   @RESERV
        MOV  @PABBUF,@PRMZAL Adresse des String-Buffer's im VDP-RAM sichern
        A    @D128,@PRMZAL   Der Buffer liegt 128 Bytes hinter dem PAB-Buffer
        MOV  @PAB,RØ       Adresse des Pab nach RØ
        A    @DØ5,RØ       F}nf hinzuaddieren
        MOV  RØ,@PAB5      Adresse in den Buffer schieben
        BL   @INOUT        Datei er!ffnen
        MOVB @WRITEE,R1    Ausgabe-Befehl in's PAB bringen
        MOV  @PAB,RØ
        BLWP @VSBW
        BL   @INOUT        Datei auf Ausgabe {ndern
        LI   R1,>Ø1ØØ      Ein Byte zum Drucker schicken
        MOV  @PAB5,RØ
        BLWP @VSBW
        MOV  @PABBUF,RØ
        LI   R1,>Ø8ØØ      Grafikcode in den PAB-Buffer schreiben
        BLWP @VSBW
        BL   @INOUT        Drucker auf Grafik umstellen
        LI   R1,>Ø5ØØ      F}nf Bytes zum Drucker schicken
        MOV  @PAB5,RØ
        BLWP @VSBW
        MOV  @PABBUF,RØ    Die 1. H{lfte ab Position 6B drucken
        LI   R1,MYDAT1
        LI   R2,5
        BLWP @VMBW
        BL   @INOUT        Daten zum Drucker schicken
        LI   R1Ø,2B        Druckzeilen-Z{hler
        LI   RB,ZLDATA     Adresse der Zeilenbearbeitungsdaten
HCOPY1  LI   R9,2          Eine Druckzeile besteht aus zwei H{lften
        MOV  *RB+,R13      1.Adresse nach R13
        MOV  *RB+,R14      2.Adresse nach R14
        MOV  *RB+,@HELP1   Derzeitige Variablen-Adresse nach HELP1
        MOVB *R8+,@HELP2   1.Schleifenz{hler nach HELP2
        MOVB *R8+,@HELP3   2.Schleifenz{hler nach HELP3
        LI   R11,PATT      Adresse des Pat-Buffer's laden
        A    R11,R13       Konstante hinzuaddieren
        A    R11,R14
HCOPY2  CLR  R15           R15=Ø
HCOPY3  MOV  R13,R1        CPU-Adresse nach R1
        A    @DØB,R13      Adresse um Stringl{nge erh!hen
        MOV  @PRMZAL,RØ    VDP-Bufferadresse nach RØ
        LI   R2,B          Acht Bytes sind zu }bertragen
        BLWP @VMBW         String sichern
        LI   R1,STRIN1     CPU-Adresse des String laden
        BLWP @VMBR         String in's Ziel bringen
        MOV  R14,R1        CPU-Adresse nach R1
        A    @DØB,R14      Adresse um Stringl{nge erh!hen
        BLWP @VMBW         String sichern
        LI   R1,STRIN2     CPU-Adresse des String laden
        BLWP @VMBR         String in's Ziel bringen
*
        LWPI MYREG2        2.Satz Arbeitsregister laden
        LI   R15,MATRIX    R15 mit der Adresse der Bitmatrix laden
        LI   R2,56         56 Bytes sind zu l!schen
HCOPY4  MOVB @BYTØ,*R15+   Byte l!schen und Adresse plus eins
        DEC  R2            Alle Bytes gel!scht ?
        JNE  HCOPY4        Wenn nicht, n{chstes Byte
*
        CLR  R1            R1=Ø
        LI   R15,MATRIX    Bitmatrixadresse erneuern
        MOV  @HELP1,R14    Aktuelle STRIN1- oder STRIN2-Adresse gewinnen
        LI   R13,STRIN2    R13 mit STRIN2-Adresse laden
        MOVB @HELP2,R9     Ersten SChleifenz{hlerwert laden
        MOVB R9,R9         Soll die Schleife ausgef}hrt werden ?
        JEQ  HCOPY6        Nein, also n{chste Schleife
*
        SWPB R9            Ja, also Wort in Byte verwandeln
        MOVB R1,R9         MS-Byte von R9 l!schen
HCOPY5  LI   RØ,7          RØ mit Verschiebewert laden
        MOVB *R14+,R5      Byte nach R5 kopieren
        MOVB R5,R7         Byte in R7 sichern
        BL   @COPIER       Byte in Bitmatrix eintragen
        DEC  R9            Schleifenz{hler um eins erniedrigen
        JNE  HCOPY5        Wenn Schleife noch nicht fertig, n{chstes Byte
*
HCOPY6  MOVB @HELP3,R9     Zweiten Schleifenz{hlerwert laden
        MOVB R9,R9         Soll die Schleife ausgef}hrt werden ?
        JEQ  HCOPY8        Nein, also Bitmatrix auswerten
*
        SWPB R9            Ja, also Wort in Byte verwandeln
        MOVB R1,R9         MS-Byte von R9 l!schen
HCOPY7  LI   RØ,7          RØ mit Verschiebewert laden
        MOVB *R13+,R5      Byte nach R5 kopieren
        MOVB R5,R7         Byte in R7 sichern
        BL   @COPIER       Byte in Bitmatrix eintragen
        DEC  R9            Schleifenz{hler um eins erniedrigen
        JNE  HCOPY7        Wenn Schleife noch nicht fertig, n{chstes Byte
*
HCOPY8  BL   @DSGNCH       Bitmatrix auswerten
        INC  R15           Spaltenz{hler um eins erh!hen
        CI   R15,16        Eine Zeilenh{lfte ausgewertet ?
        JNE  HCOPY3        Nein, also n{chste Spalte
*
        DEC  R9            Eine komplette Zeile ausgewertet ?
        JNE  HCOPY9        Wenn nicht, erste H{lfte drucken
*
        BL   @DRUCK2       Sonst zweite H{lfte drucken
        JMP  HCOPYA        N{chste Zeile auswerten
*
HCOPY9  BL   @DRUCK1       Erste H{lfte drucken
        JMP  HCOPY2        Zweite h{lfte auswerten
*
HCOPYA  DEC  R1Ø           Druckzeilenz{hler um eins erniedrigen
        JNE  HCOPY1        Noch nicht alle Zeilen gedruckt
*
        LI   R1,>Ø5ØØ      Alle Zeilen gedruckt, f}nf Bytes zum Drucker
        MOV  @PAB5,RØ
        BLWP @VSBW
        MOV  @PABBUF,RØ    Daten f}r Zeilenvorschub laden
```

```
       LI   R1,MYDAT3
       LI   R2,5
       BLWP @VMBW
       BL   @INOUT        Zeilenvorschub ausf}hren
       LI   R1,>Ø1ØØ      Ein Byte zum Drucker schicken
       MOV  @PAB5,RØ
       BLWP @VSBW
       MOV  @PABBUF,RØ
       LI   R1,>ØFØØ      Textmoduscode in den PAB-Buffer schreiben
       BLWP @VSBW
       BL   @INOUT        Drucker wieder auf Standardmodus umschalten
       CLR  R1            Aus PAB-Buffer kein Byte mehr zum Drucker schicken
       MOV  @PAB5,RØ
       BLWP @VSBW
       B    @SHUT         Datei schlie~en und n{chste Basic-Anweisung
*
* CALL LINK("SAVE","DSK"(1-3)"."(NAME))
*
* Dieses Programm erlaubt es, die Grafic im Variable BØ - Format auf
* Diskette abzuspeichern.
*
SAVE   LWPI MYREGI        Eigene Arbeitsregister laden
       LI   R15,PDATA       Adresse der aktuellen PAB-Daten laden
       MOV  R15,@BUFADR     Adresse sichern
       MOV  @D1Ø5,@>83ØC    1Ø5 Bytes sollen im VDP-RAM reserviert werden
       LI   R2,1Ø           Die Pabdaten sind zehn Bytes lang
       MOV  @D25,R5
       BL   @RESERV         Platz f}r den Pab-Buffer im VDP-RAM reservieren
       LI   R1,>12ØØ      Display/Variable/Output
       MOV  @PAB,RØ       In RØ steht die Adresse von Pab
       INC  RØ            RØ=RØ+1
       BLWP @VSBW         Byte in's VDP-RAM bringen
       BL   @GETSTR       Datei-Namen }bernehmen und in's VDP-RAM bringen
       BL   @INOUT        Datei er|ffnen
       LI   R1,>Ø3ØØ      Byte f}r Write
       MOV  @PAB,RØ       Pab-Adresse laden
       BLWP @VSBW         Byte in's VDP-RAM bringen
       BL   @INOUT        Datei-Operation durchf}hren
       LI   R14,PATT      R14 mit Adresse des P.D.T.-Buffer's laden
       MOV  @PAB,RØ       In PAB+5 steht die Anzahl der Zeichen f}r Write
       A    @DØ5,RØ
       LI   R1,BØ         BØ Bytes sollen bei Write gespeichert werden
       SWPB R1            Wort in Byte verwandeln
       BLWP @VSBW         Ein Byte in's VDP-RAM schreiben
       MOV  @PABBUF,RØ    In RØ steht die Adresse des Pab-Buffers im VDP-RAM
       LI   R2,BØ         BØ Bytes m}~en jedesmal in den Pab-Buffer
       LI   R12,153       153 mal BØ Bytes m}~en gespeichert werden
SAVE1  MOV  R14,R1        Aktuelle Buffer-Adresse nach R1
       BLWP @VMBW         Bytes in den Pab-Buffer bringen
       A    @DBØ,R14      Adressenkonstante hinzuaddieren
       BL   @INOUT        Datei-Operation durchf}hren
       DEC  R12           Sind die 153 Durchg{nge ausgef}hrt ?
       JNE  SAVE1         Wenn nicht, n{chster Durchgang
*
       MOV  @PAB,RØ       Jetzt m}~en noch 64 Bytes gespeichert werden
       A    @DØ5,RØ
       LI   R1,4B         R1=4B
       SWPB R1            Wort in Byte verwandeln
       BLWP @VSBW         Byte in's VDP-RAM bringen
       MOV  @PABBUF,RØ    In RØ steht die Adresse des Pab-Buffer's
       MOV  R14,R1        Wieder die aktuelle Buffer-Adresse nach R1
       LI   R2,48         Die restlichen 4B Bytes in den Pab-Buffer bringen
       BLWP @VMBW         Bytes in's VDP-RAM bringen
       BL   @INOUT        Datei-Operation durchf}hren
       B    @SHUT         Datei schlie~en und zur}ck in's Basic
*
* CALL LINK("OLD","DSK"(1-3)"."(NAME))
*
* Dieses Programm erlaubt es, die auf Diskette gespeicherten Grafiken
* zu laden.
*
OLD    LWPI MYREGI        Eigene Arbeitsregister laden
       LI   R15,PDATA       Adresse der aktuellen PAB-Daten laden
       MOV  R15,@BUFADR     Adresse sichern
       MOV  @D1Ø5,@>B3ØC    1Ø5 Bytes sollen im VDP-RAM reserviert werden
       LI   R2,1Ø           Die Pabdaten sind zehn Bytes lang
       MOV  @D25,R5
       BL   @RESERV         Platz f}r den Pab-Buffer im VDP-RAM reservieren
       LI   R1,>14ØØ      Display/Variable/Input
       MOV  @PAB,RØ       In RØ steht die Adresse des Pab
       INC  RØ            RØ=RØ+1
       BLWP @VSBW         Byte in's VDP-RAM bringen
       BL   @GETSTR       Datei-Namen }bernehmen und in's VDP-RAM bringen
       BL   @INOUT        Datei er|ffnen
       LI   R1,>Ø2ØØ      Byte f}r Read
       MOV  @PAB,RØ       Pab-Adresse laden
       BLWP @VSBW         Byte in's VDP-RAM bringen
       BL   @INOUT        Datei-Operation durchf}hren
       LI   R14,PATT      R14 mit Adresse des P.D.T.-Buffer's laden
       MOV  @PABBUF,RØ    In RØ steht die Adresse des P.D.T.-Buffer's
       LI   R2,8Ø         BØ Bytes m}~en aus dem Pab-Buffer gelesen werden
       LI   R12,153       153 mal BØ Bytes m}~en geladen werden
OLD1   BL   @INOUT        Datei-Operation durchf}hren
       MOV  R14,R1        Aktuelle Buffer-Adresse nach R1
       BLWP @VMBR         Bytes aus dem Pab-Buffer sichern
       A    @D8Ø,R14      Adressenkonstante hinzuaddieren
       DEC  R12           Sind die 153 Durchg{nge durchgef}hrt ?
       JNE  OLD1          Wenn nicht, n{chster Durchgang
*
       BL   @INOUT        Datei-Operation durchf}hren
       MOV  R14,R1        Wieder aktuelle Buffer-Adresse nach R1
       LI   R2,48         Die restlichen 48 Bytes aus dem Pab-Buffer lesen
       MOV  @PABBUF,RØ    In RØ steht die Adresse des Pab-Buffer's
       BLWP @VMBR         Bytes aus dem VDP-RAM lesen
       B    @SHUT
*
* Ende Segment "@BITMAP3S"
*
```

```
* Segment "@BITMAP4S"
*
* *****************
* *               *
* * Hauptprogramme *
* *               *
* *     2.Teil    *
* *               *
* *****************
*
* CALL LINK("CKREIS",Y,X,R)
*
* Dieses Programm erlaubt es, durch }bergabe der Mittelpunkt-Koordinaten
* und des Radius Kreise zu l|schen.
*
CKREIS LIMI Ø               VDP-Interrupt unterdr}cken
*
       LWPI MYREGI          Eigene Arbeitsregister laden
       MOVB @BYTØ,@P        Pixel l|schen
       JMP  KREIS1          Sprung in's Hauptprogramm
*
* CALL LINK("KREIS",Y,X,R)
*
* Dieses Programm erlaubt es, durch }bergabe der Mittelpunkt-Koordinaten
* und des Radius Kreise zu erzeugen.
*
KREIS  LIMI Ø               VDP-Interrupt unterdr}cken
*
       LWPI MYREGI          Eigene Arbeitsregister laden
       MOVB @BYT1,@P        Pixel setzen
KREIS1 LI   R1,1            1.Parameter
       BL   @TSTGR          Parameter }bernehmen und testen
       DATA 1,192
       MOV  @FAC,@Y         Parameter ist Y-Koordinate
       INC  R1              2.Parameter
       BL   @TSTGR          Parameter }bernehmen und testen
       DATA 1,256
       MOV  @FAC,@X         Parameter ist X-Koordinate
       INC  R1              3.Parameter
       BL   @TSTGR          Parameter }bernehmen und testen
       DATA 1,255
       MOV  @FAC,R9         Parameter ist Radius in Bildschirmpixel
       MOV  @FAC,R5         Radius ist jetzt in R5 und R9
       MPY  R9,R5           Den Radius quadrieren
       MOV  R6,@RQ          Ergebnis in RQ sichern
       CLR  R7              R7=Ø
KREIS2 MOV  R7,R5           R7=R5 , R7 ist gleichzeitig Schleifenz{hler
       MPY  R7,R5           Schleifenz{hler quadrieren
       MOV  @RQ,R8          R8=RQ
       S    R6,R8           R8=R8-R6
       MOV  R8,@FAC         R8 nach FAC bringen
*
       LWPI MYREG2          2.Satz Arbeitsregister laden
       BL   @SQUARE         Wurzel aus FAC und Ergebnis nach FAC
*
       LWPI MYREGI          Eigene Arbeitsregister wieder laden
       MOV  @FAC,R8         Ergebnis zur}ck nach R8
       MOV  @X,R11          R11=X-Koordinate
       S    R7,R11          R11=R11-R7
       MOV  R11,@SATZ+2     Ergebnis an richtige Position bringen
       MOV  R11,@SATZ+6
       MOV  @Y,R11          R11=Y-Koordinate
       A    R8,R11          R11=R11+R8
       MOV  R11,@SATZ
       MOV  R11,@SATZ+8
       MOV  @X,R11          R11=X-Koordinate
       A    R7,R11          R11=R11+R7
       MOV  R11,@SATZ+1Ø
       MOV  R11,@SATZ+14
       MOV  @Y,R11          R11=Y-Koordinate
       S    R8,R11          R11=R11-R8
       MOV  R11,@SATZ+4
       MOV  R11,@SATZ+12
       MOV  @X,R11          R11=X-Koordinate
       S    R8,R11          R11=R11-R8
       MOV  R11,@SATZ+18
       MOV  R11,@SATZ+22
       MOV  @Y,R11          R11=Y-Koordinate
       A    R7,R11          R11=R11+R7
       MOV  R11,@SATZ+16
       MOV  R11,@SATZ+24
       MOV  @X,R11          R11=X-Koordinate
       A    R8,R11          R11=R11+R8
       MOV  R11,@SATZ+26
       MOV  R11,@SATZ+3Ø
       MOV  @Y,R11          R11=Y-Koordinate
       S    R7,R11          R11=R11-R7
       MOV  R11,@SATZ+2Ø
       MOV  R11,@SATZ+28
*
       LWPI MYREG2          Zweiten Satz Arbeitsregister laden
       MOV  @D32,@PRMZAL    32 Koordinaten sollen bearbeitet werden
       MOVB @BYT1,R7        Die Koordinaten kommen von au~erhalb
       BL   @SETTO          Koordinaten setzen
*
       LWPI MYREGI          Normale Arbeitsregister wieder laden
       INC  R7              Schleifenz{hler um eins erh|hen
       DEC  R9              Wert des Radius um eins verringern
       JNE  KREIS2          Wenn der Radius noch nicht null, n{chster Durchgang
*
       LIMI 2               VDP-Interrupt wieder zulassen
       B    @BASIC          N{chste Basic-Anweisung
*
* CALL LINK("CWLINE",Y,X,L{nge,Winkel zur Horizontalen)
*
* Dieses Programm erlaubt es, durch Angabe eines Punktes, der L{nge der
* Linie und eines Winkels zur Horizontalen Linien zu l|schen.
* (Positive Winkel drehen im Urzeigersinn)
*
CWLINE LIMI Ø               VDP-Interrupt unterdr}cken
*
       LWPI MYREGI          Eigene Arbeitsregister laden
       MOVB @BYTØ,@P        Pixel l|schen
       JMP  LINE1           Sprung in's Hauptprogramm
*
* CALL LINK("WLINE",Y,X,L{nge,Winkel zur Horizontalen)
*
* Dieses Programm erlaubt es, durch Angabe eines Punktes, der L{nge der
```

```
* Linie und eines Winkels zur Horizontalen Linien zu erzeugen.
* (Positive Winkel drehen im Urzeigersinn)
*
WLINE  LIMI Ø               VDP-Interrupt unterdr}cken
*
       LWPI MYREGI          Eigene Arbeitsregister laden
       MOVB @BYT1,@P        Pixel setzen
LINE1  LI   R1,1            1.Parameter
       BL   @TSTGR          Parameter }bernehmen und testen
       DATA 1,192
       MOV  @FAC,@YA        Y-Koordinate des Anfangspunktes nach YA
       INC  R1              2.Parameter
       BL   @TSTGR          Parameter }bernehmen und testen
       DATA 1,256
       MOV  @FAC,@XA        X-Koordinate des Anfangspunktes nach XA
       INC  R1              3.Parameter
       BL   @TSTGR          Parameter }bernehmen und testen
       DATA 1,32Ø
       BLWP @XMLLNK         L{nge der Linie in Gleitkomma verwandeln
       DATA CIF
       BL   @TRANS          Gleitkomma-Wert sichern
       DATA FAC,FPBUF1
       INC  R1              4.Parameter
       BL   @TSTGR          Parameter }bernehmen und testen
       DATA Ø,359
       BLWP @XMLLNK         Winkel in Gleitkomma verwandeln
       DATA CIF
       BL   @TRANS          Gleitkomma-Wert sichern
       DATA FAC,ARG
       BL   @TRANS          Divisions-Wert nach FAC
       DATA DIVISO,FAC
       BLWP @XMLLNK         Winkel in Radiant umwandeln
       DATA FDIV
       BL   @TRANS          Radiant-Winkel sichern
       DATA FAC,WINKEL
       MOVB @BYTØ,@GPLSTA   GPL-Statusbyte l|schen
       LWPI GPLWS           GPL-Arbeitsregister laden
       BLWP @GPLLNK         Cosinus des Winkels berechnen
       DATA COS
*
       LWPI MYREGI          Normale Arbeitsregister wieder laden
       BL   @TRANS          L{nge der Linie nach ARG bringen
       DATA FPBUF1,ARG
       BLWP @XMLLNK         Cosinus des Winkels und L{nge multiplizieren
       DATA FMULT
       BLWP @XMLLNK         Ergebnis in Integer verwandeln
       DATA CFI
       MOV  @FAC,@XE        Ergebnis ist X-Koordinate des Endpunktes
       BL   @TRANS          Winkel wieder nach FAC bringen
       DATA WINKEL,FAC
       MOVB @BYTØ,@GPLSTA   GPL-Statusbyte l|schen
       LWPI GPLWS           GPL-Arbeitsregister laden
       BLWP @GPLLNK         Sinus des Winkels berechnen
       DATA SIN
*
       LWPI MYREGI          Normale Arbeitsregister wieder laden
       BL   @TRANS          L{nge der Linie nach ARG
       DATA FPBUF1,ARG
       BLWP @XMLLNK         Sinus des Winkels und l{nge multiplizieren
       DATA FMULT
       BLWP @XMLLNK         Ergebnis in Integer umwandeln
       DATA CFI
       MOV  @FAC,@YE        Ergebnis ist Y-Koordinate des Endpunktes
       A    @XA,@XE         X-Koordinate des Endpunktes vervollst{ndigen
       A    @YA,@YE         Y-Koordinate des Endpunktes vervollst{ndigen
       JMP  LINE3           Sprung in's Hauptprogramm
*
* CALL LINK("CLINE",Y1,X1,Y2,X2)
*
* Dieses Programm erlaubt es, zwei angegebene Punkte mittels einer Linie
* zu verbinden. Bei diesem Programm wird die Linie gel|scht.
*
CLINE  LIMI Ø               VDP-Interrupt unterdr}cken
       LWPI MYREGI          Eigene Arbeitsregister laden
       MOVB @BYTØ,@P        Pixel l|schen
       JMP  LINE2           Sprung zur Koordinaten-}bernahme
*
* CALL LINK("LINE",Y1,X1,Y2,X2)
*
* Dieses Programm erlaubt es, zwei angegebene Punkte mittels einer Linie
* zu verbinden. Bei diesem Programm wird die Linie erzeugt.
*
LINE   LIMI Ø               VDP-Interrupt unterdr}cken
*
       LWPI MYREGI          Eigene Arbeitsregister laden
       MOVB @BYT1,@P        Pixel setzen
LINE2  LI   R1,1            1.Parameter
       BL   @TSTGR          Parameter }bernehmen und testen
       DATA 1,192
       MOV  @FAC,@YA        Y-Koordinate des Anfangspunktes sichern
       INC  R1              2.Parameter
       BL   @TSTGR          Parameter }bernehmen und testen
       DATA 1,256
       MOV  @FAC,@XA        X-Koordinate des Anfangspunktes sichern
       INC  R1              3.Parameter
       BL   @TSTGR          Parameter }bernehmen und testen
       DATA 1,192
       MOV  @FAC,@YE        Y-Koordinate des Endpunktes sichern
       INC  R1              4.Parameter
       BL   @TSTGR          Parameter }bernehmen und testen
       DATA 1,256
       MOV  @FAC,@XE        X-Koordinate des Endpunktes sichern
LINE3  MOV  @YA,@FAC        Y-Anfangskoordinate nach FAC
       BLWP @XMLLNK         Wert in Gleitkomma umwandeln
       DATA CIF
       BL   @TRANS          Gleitkomma-Wert sichern
       DATA FAC,YAFP
       MOV  @XA,@FAC        X-Anfangskoordinate nach FAC
       BLWP @XMLLNK         Wert in Gleitkomma umwandeln
       DATA CIF
       BL   @TRANS          Gleitkomma-Wert sichern
       DATA FAC,XAFP
       MOV  @YE,@FAC        Y-Endkoordinate nach FAC
       BLWP @XMLLNK         Wert in Gleitkomma umwandeln
       DATA CIF
       BL   @TRANS          Gleitkomma-Wert sichern
       DATA FAC,YEFP
       MOV  @XE,@FAC        X-Endkoordinate nach FAC
```

```
        BLWP @XMLLNK            Wert in Gleitkomma umwandeln
        DATA CIF
        BL   @TRANS             Gleitkomma-Wert sichern
        DATA FAC,XEFP
*
        LWPI MYREG2             Zweiten Satz Arbeitsregister laden
        MOV  @DØ2,@PRMZAL       Zwei Koordinaten sollen bearbeitet werden
        MOVB @BYT1,R7           Die Koordinaten kommen von au~erhalb
        MOV  @YA,@SATZ          Y-Anfangskoordinate nach SATZ
        MOV  @XA,@SATZ+2        X-Anfangskoordinate nach SATZ plus zwei
        BL   @SETTO             Koordinaten bearbeiten
*
        LWPI MYREG1             Normale Arbeitsregister wieder laden
        MOV  @XA,R2             Hier werden die Anfangs- und Endkkordinaten in
        MOV  @YA,R3             Registern gesichert. Zur ]bersichtlichkeit be-
        MOV  @XE,R4             nutze ich jetzt nur noch die Variablen zur Er-
        MOV  @YE,R5             kl{rung.
        C    R2,R4              Ist XA=XE ?
        JNE  LINE4              Wenn nicht, Sprung
*
        C    R3,R5              Wenn doch, ist auch YA=YE ?
        JNE  LINE4              Wenn nicht, Sprung
*
        LIMI 2                  Wenn doch, VDP-Interrupt wieder zulassen
        B    @BASIC             und n{chste Basic-Anweisung
LINE4   S    R2,R4              R4=XE-XA
        S    R3,R5              R5=YE-YA
        MOV  R4,R13             R13=XE-XA
        MOV  R5,R15             R15=YE-YA
        ABS  R4                 ABS(XE-XA)
        ABS  R5                 ABS(YE-YA)
        C    R4,R5              Ist R4>R5 ?
        JGT  LINE5              Wenn ja, Sprung
*
        JMP  LINE6              Sonst Sprung
*
LINE5   B    @LINEB
LINE6   MOV  R13,@FAC           XE-XA steht jetzt in FAC
        BLWP @XMLLNK            Wert in Gleitkomma umwandeln
        DATA CIF
        BL   @TRANS             Den FAC-Wert nach ARG kopieren
        DATA FAC,ARG
        MOV  R5,@FAC            ABS(YE-YA) nach FAC
        BLWP @XMLLNK            Wert in Gleitkomma-Wert umwandeln
        DATA CIF
        BLWP @XMLLNK            Gleitkomma-Division: ARG/FAC
        DATA FDIV
        BL   @TRANS             Ergebnis sichern
        DATA FAC,FPBUF1
        MOV  @YA,R8             R8=YA
        MOV  @YE,R9             R9=YE
        S    R9,R8              R8=YA-YE
        ABS  R8                 ABS(YA-YE)
        INC  R8                 R8=R8+1
        CI   R15,Ø              Ist R15=Ø ?
        JGT  LINE9              Wenn R15 gr|~er ist, Sprung
*
LINE7   DEC  @YA                YA=YA-1
        DEC  R8                 Ist R8=Ø ?
```

```
        JNE  LINE8              Wenn nicht, Sprung
*
        LIMI 2                  Wenn doch, VDP-Interrupt wieder zulassen
        B    @BASIC             N{chste Basic-Anweisung
LINE8   BL   @TRANS             Gleitkomma-Wert nach ARG transferieren
        DATA FPBUF1,ARG
        BL   @SUBLI1            Berechnung durchf}hren und Koordinaten setzen
        JMP  LINE7              N{chste Schleife durchf}hren
*
LINE9   INC  @YA                YA=YA+1
        DEC  R8                 Ist R8=Ø ?
        JNE  LINEA              Wenn nicht, Sprung
*
        LIMI 2                  Wenn doch, VDP-Interrupt wieder zulassen
        B    @BASIC             N{chste Basic-Anweisung
LINEA   BL   @TRANS             Gleitkomma-Wert nach ARG transferieren
        DATA FPBUF1,ARG
        BL   @SUBLI1            Berechnung durchf}hren und Koordinaten setzen
        JMP  LINE9              N{chste Schleife durchf}hren
*
LINEB   MOV  R15,@FAC           YE-YA steht jetzt in FAC
        BLWP @XMLLNK            Wert in Gleitkomma umwandeln
        DATA CIF
        BL   @TRANS             Den FAC-Wert nach ARG kopieren
        DATA FAC,ARG
        MOV  R4,@FAC            ABS(XE-XA) nach FAC
        BLWP @XMLLNK            Wert in Gleitkomma-Wert umwandeln
        DATA CIF
        BLWP @XMLLNK            Gleitkomma-Division: ARG/FAC
        DATA FDIV
        BL   @TRANS             Ergebnis sichern
        DATA FAC,FPBUF1
        MOV  @XA,R8             R8=XA
        MOV  @XE,R9             R9=XE
        S    R9,R8              R8=XA-XE
        ABS  R8                 ABS(XA-XE)
        INC  R8                 R8=R8+1
        CI   R13,Ø              Ist R13=Ø ?
        JGT  LINEE              Wenn R13 gr|~er ist, Sprung
*
LINEC   DEC  @XA                XA=XA-1
        DEC  R8                 Ist R8=Ø ?
        JNE  LINED              Wenn nicht, Sprung
*
        LIMI 2                  Wenn doch, VDP-Interrupt wieder zulassen
        B    @BASIC             N{chste Basic-Anweisung
LINED   BL   @TRANS             Gleitkomma-Wert nach ARG transferieren
        DATA FPBUF1,ARG
        BL   @SUBLI2            Berechnung durchf}hren und Koordinaten setzen
        JMP  LINEC              N{chste Schleife durchf}hren
*
LINEE   INC  @XA                XA=XA+1
        DEC  R8                 Ist R8=Ø ?
        JNE  LINEF              Wenn nicht, Sprung
*
        LIMI 2                  Wenn doch, VDP-Interrupt wieder zulassen
        B    @BASIC             N{chste Basic-Anweisung
LINEF   BL   @TRANS             Gleitkomma-Wert nach ARG transferieren
        DATA FPBUF1,ARG
```

```
       BL   @SUBLI2        Berechnung durchf}hren und Koordinaten setzen
       JMP  LINEE          N{chste Schleife durchf}hren
*
* Ende Segment "@BITMAP4S"
*
```

Fortsetzung von S. 41

BIT MAP MODE

hen alle ASCII-Zeichen von 32 bis 126 einschließlich.

CALL LINK ("CHAR", Zeile, Spalte, String, Wiederholungen (Optional))-->
Dieser Befehl entspricht dem von Basic her bekannten Befehl CALL CHAR, jedoch wirkt er nicht auf ein bestimmtes Zeichen, sondern auf eine Bildschirmposition. Der String muß 16 Zeichen lang sein. Parameter vier ist optional, d.h. daß der definierte String ab der angegebenen Position bis zu 768mal auftauchen kann. Wird der vierte Parameter nicht angegeben, so erscheint der selbstdefinierte String nur einmal auf dem Bildschirm.

CALL LINK ("COLOR", Zeile, Spalte, String, Wiederholungen (Optional))-->
Dieser Befehl entspricht im wesentlichen dem Befehl CALL LINK ("CHAR",...), jedoch haben Sie hiermit die Möglichkeit, die Farben der Grafik zu beeinflussen. Jeweils acht Pixel können eine Vordergrund- und eine Hintergrundfarbe annehmen. Dafür werden zwei Zeichen gebraucht. Da acht Pixelzeilen einen String ergeben, sind auch hierbei 16 Zeichen für eine Definition notwendig.

CALL LINK ("KREIS", Y,X,R)-->
Dieser Befehl erlaubt es, Kreise mit dem Mittelpunkt X,Y und dem Radius R zu erzeugen. Der Radius darf max. 255 betragen, jedoch kommt es bei einem Radius größer als 185 zu einer Streuung der Pixel, d.h., daß der Kreis nicht mehr geschlossen ist.

CALL LINK ("CKREIS", Y,X,R)-->
Siehe 'KREIS'.
Dieser Befehl löscht die Kreise.

CALL LINK ("LINE", Y1,X1,Y2,X2)-->
Dieser Befehl erlaubt es, Linien von der Koordinate X1,Y1 zu der Koordinate X2,Y2 zu ziehen.

CALL LINK ("CLINE", Y1,X1,Y2,X2)-->
Siehe 'LINE'.
Dieser Befehl löscht die Linien.

CALL LINK ("WLINE", Y,X,Länge, Winkel zur Horizontalen)-->
Dieser Befehl erlaubt es, Linien von der Koordinate X,Y, mit einer maximalen Länge von 320 Pixel und einem Winkel von 0 bis 359 Grad zu ziehen. Der Winkel dreht dabei im Uhrzeigersinn.

CALL LINK ("CWLINE", Y,X,Länge, Winkel zur Horizontalen)-->
Siehe 'WLINE'.
Dieser Befehl löscht die Linien.

CALL LINK ("SAVE", "Dateiname")-->
Dieser Befehl erlaubt es, die mittels CALL LINK ("S

Dieser Befehl bietet die Möglichkeit, eine Grafik auf Diskette abzuspeichern.

CALL LINK ("OLD", "Dateiname")-->
Dieser Befehl erlaubt es, die mittels CALL LINK ("SAVE",...) abgespeicherte Grafik wieder zu laden.

CALL LINK ("HCOPY")-->
Mit diesem Befehl können Hardcopys der Grafik auf einem Seikosha GP 100 A über die PIO der RS 232-Karte angefertigt werden.

CALL LINK ("ZEIG", Sekunden)-->
Dieser Befehl bringt für eine Zeit von maximal 480 Sekunden die Grafik auf den Bildschirm und kehrt danach zum Titelbildschirm zurück.

Abschließend sei noch gesagt, daß die Pixelzeilen 1 bis 192 und die Pixelspalten 1 bis 256 erlaubt sind. Nur die Befehle, wo Zeile und Spalte steht, verlangen die normalen Zeilen- und Spaltenwerte. Auch steht, wie Sie wohl bereits gemerkt haben werden, die Y-Koordinate an erster Stelle und nicht wie mathematisch richtig, zuerst die X-Koordinate. Da wir aber sonst auch immer erst die Zeile und dann die Spalte eingeben, erschien mir diese Anordnung besser.
Und nun wünsche ich Ihnen viel Spaß beim Eintippen.

Bernd Bertling

Anm. d. Red.: Der Bit-Map-Mode ist bekanntlich mit Basic nicht richtig in Einklang zu bringen, deshalb hat der Autor hier die Lösung gewählt, daß das Bild „blind" aufgebaut wird und erst am Schluß für eine bestimmte Zeit gezeigt wird. Das Basic-Programm muß daher immer, bevor CALL LINK ("ZEIG",X) ausgeführt wird, abgespeichert werden, da anschließend zum Titelbild zurückgeführt wird und damit das im Speicher befindliche Basicprogramm verloren ist.



ASSEMBLER IN TI-BASIC

Der Assembler gliedert sich in drei Teile.
Teil -1-, der Editor, steuert alle erforderlichen Eingaben. Der Editor stellt verschiedene Unterprogramme zur Verfügung, um eine Quelldatei einzugeben und zu bearbeiten. Es werden einige Hilfen gegeben, um die Arbeit mit dem Assembler zu erleichtern.
Teil -2-, der eigentliche Assembler, verarbeitet neben den 64 auf der TI-Konsole möglichen Mnemonic und den verschiedenen Adressierungsarten einige Direktiven sowie Pseudo-Instruktionen.
Teil -3- umfaßt verschiedene Unterprogramme, die Umrechnungs- und Berechnungsroutinen, die Fehlerausgabe und Kontrollroutinen enthalten
Das gesamte Programm nimmt 8440 Bytes des Speichers in Anspruch. Nach dem Start des Programmes stehen im TI-Basic etwas über 4 K-Byte für die Quelldatei zur Verfügung.
Nach einigen Änderungen ist das Programm auch fürs EX-Basic mit Speichererweiterung geeignet. Die Änderungen umfassen insbesondere die Umrechnungsroutinen zwischen Dezimal- und Hexadezimalzahlen. Ein Listing zeigt die erforderlichen Änderungen. Die Fähigkeiten des EX-Basic sind aus Gründen der Kompatibilität nicht genutzt.
So kann die Anzahl der Label und der Umfang der Quelldatei verdoppelt werden; eine Straffung des Programmes und die Beschleunigung des Assemblerdurchlaufes sind möglich.

Zur Eingabe:
Grundsätzlich erfolgen alle Eingaben als Stringvariable. Es stehen bei der Eingabe alle Möglichkeiten des TI-Basic zur Verfügung, innerhalb des Strings stehen die Editiermöglichkeiten des TI-Basic zur Wahl.
Durch diese Eingabe müssen einige Besonderheiten beachtet werden.
Eintragungen in die Quelldatei dürfen kein Komma enthalten (ASCII-Zeichen 44).
Abschließende Leerzeichen (ASCII-Zeichen 32) sind nicht möglich, da sie vom TI-Basic gestrichen werden.
Änderungen in der Quelldatei sind nur durch Neueingabe des Datensatzes möglich.
Bei der Eingabe ist das Format des Datensatzes vorgeschrieben. Das Format muß eingehalten werden, da sonst eine Fehlermeldung gegeben ist.
Die Länge eines Datensatzes darf zunächst nicht 40 Zeichen überschreiten. Die Fehler des Satzes sind genau bestimmt.
Beispiel für das Format:

```
100 LB MOVB
@<834A(R14).R0
12345678901234567890
         1         2
12345678901234567890
         3         4
```

Die Zeichen 4, 7, 12 müssen <Space>sein (ASCII-Zeichen 32).
Zeichen 1 bis 3 enthalten die Zeilennummer.
Zeichen 5 und 6 umfassen das Kennzeichenfeld (für Label)
Zeichen 8 bis 11 stellen das Operationsfeld dar.
Ab Zeichen 13 beginnt das Operandenfeld.
Ein Kommentarfeld ist nicht vorgesehen.

Zeilennummer:
Der Wert der Nummer muß zwischen 100 und 999 liegen und muß ganzzahlig sein. Die Sätze innerhalb der Quelldatei werden an Hand der Zeilennummer sortiert.

Label:
Die Länge des Label ist auf zwei Zeichen begrenzt. Das erste Zeichen muß alphabetisch sein. Das zweite Zeichen ist beliebig. Wird ein Label mit einem Buchstaben, gefolgt von <Space> benutzt, so kann <Space> im Operandenfeld weggelassen werden.
Wird kein Label gewünscht, so sind zwei Leerzeichen einzugeben. Die Quelldatei darf nur insgesamt 30 Label enthalten.

Operationsfeld:
Hier stehen ein Mnemonic, eine Pseudo-Instruktion oder eine Direktive. Diese müssen 1 bis 4 Zeichen lang sein.
Eine Kontrolle auf Gültigkeit erfolgt, wie bei den Labeln und den Operanden, erst beim Assemblerdurchlauf.

Operandenfeld:
Es darf kein <Space> innerhalb des Operandenfeldes stehen.
Wird ein Register adressiert, so muß vor der Registerzahl immer ein "R" stehen.
Es sind nur eindeutige Ausdrücke erlaubt.
Konstanten können als Dezimalzahlen (0 – 65535), als negative Dezimalzahlen (–1 bis –32768) oder als Hexadezimalzahlen (>0000 bis >FFFF) eingegeben werden.
Hexadezimalzahlen haben vorne ein 'größer als' Zeichen (">") und sind 1–4 Zeichen lang. Führende Nullen können weggelassen werden.
Symbole, mit denen auf Labels Bezug genommen wird, dürfen nur zwei Zeichen lang sein. Die Zahl der Symbole ist auf 40 begrenzt.
Der vorliegende Assembler weist zusätzlich die Möglichkeit auf, bestimmte Systemadressen durch Nennen des Namens zu benutzen. Hierbei handelt es sich um wichtige Schreib- und Leseadressen sowie BLWP-Vektoren. Die Adressen werden berechnet, wenn der Systemadressenname mit vorangehendem "at"-Zeichen (-@-) im Operandenfeld steht.

Im Basic stehen 30, im EX-Basic 42 Systemadressen so zur Verfügung (siehe Liste).

Der Vorteil ist, daß nicht die Adresse, sondern der wesentlich einfacher zu merkende Name im Programm steht, wodurch auch die Lesbarkeit eines Programmes erhöht wird.
Beispiele:

```
160 BF BLWP @VSBW
220 MOVB @GRMRD.R0
```

Beim Eingeben eines Datensatzes wird zunächst nur die Zeilennummer auf Gültigkeit, die Einhaltung des Formates sowie das Operationsfeld geprüft.
Ob ein Label, ein Mnemonic oder Operand gültig ist, überprüft erst der Assembler beim assemblieren. Der Assembler gibt eventuelle Fehlermeldungen zur Fehlerroutine. Der Assemblerdurchlauf wird gestoppt, die Fehlermeldungen ausgegeben und dann zur Eingabezeile verzweigt.
Der Fehler muß, um einen erneuten Abbruch zu verhindern, vor dem nächsten Assemblerdurchlauf beseitigt werden.

ASSEMBLER

Neben der Eingabe von Datensätzen können die Unterprogramme des Editors angewählt werden. Dies geschieht durch die Eingabe eines Zeichens gefolgt von <ENTER>. Der Editor stellt folgende Unterprogramme zur Verfügung:

"N":
Die Anzahl der Sätze wird auf -0- gesetzt und der Bildschirm gelöscht

"H":
Der Bildschirm wird gelöscht.

"A":
Verzweigung zum Assembler. Erst wird geprüft, ob die erste Anweisung "AORG" und die letzte "END" ist. Wenn nicht, wird mit Fehlermeldung abgebrochen. Dann werden einige Variablen gesetzt (eventuell vorhandene Label eines vorhergehenden Assemblerdurchlaufes werden u.a. gelöscht) und zum Assembler gesprungen.
Der Assemblerdurchlauf kann durch Drücken der <ENTER>-Taste unterbrochen werden. Das Programm verzweigt dann zurück zur Eingabezeile.

"M":
Zeigt die Anzahl der Datensätze sowie die noch freien an. Nach einem Assemblerdurchlauf werden die benutzten Label und die dazugehörenden Adressen ausgegeben, um eine Verknüpfung zweier Programme zu ermöglichen.
Dieses Unterprogramm wird nach jedem fehlerfreien Assemblerdurchlauf durchgeführt.

"S":
Nach der Verzweigung wird die Eingabe einer Adresse erwartet (Dezimal oder Hexadezimal). Ab dieser Adresse werden die Speicherinhalte (Worte) in hexadezimaler Schreibweise ausgegeben. Wird die <Space>-Taste gedrückt, springt das Programm zur Eingabezeile zurück.

"L":
Dieses Programm listet aus der Quelldatei. Folgende Eingaben sind nach Verzweigung möglich:
"A"-<ENTER>: Listet ab der Zeilennummer.
Es werden zunächst 21 Sätze ausgegeben. Wird das Ende der Quelldatei erreicht, springt die Kontrolle zur Eingabezeile zurück.
Ist das Ende nicht erreicht, wird ein weiteres Kommande erwartet:
Taste -<ENTER>- springt zur Eingabezeile zurück.
Taste -<Space>- listet nur den nächsten Datensatz.
Jede andere Taste führt dazu, daß wiederum 21 Sätze ausgegeben werden.

"R":
Mit dieser Unterroutine kann ein Maschinenspracheprogramm gestartet werden. Nach der Verzweigung wird die Eingabe des Programmnamens erwartet. Dieser muß in der DEF-Table des entsprechenden Modules (Minimem bzw. Erweiterung) stehen. Sollen Parameter übergeben werden, so muß die entsprechende Basic-Zeile (1230) geändert werden.
Nach <ENTER> wird zum Maschinensprache-Programm verzweigt (mittels "LINK").
Steht der Programmname nicht in der DEF-Table kommt es zu einer Basic-Fehlermeldung mit Programmabbruch des Assemblers!

".":
Nach der Eingabe einer Dezimalzahl (0–65535) wird die Zahl in hexadezimaler Schreibweise ausgegeben. Dezimalzahlen über 32767 werden zusätzlich als "Zweier-Komplement" ausgegeben.

"-":
Nach der Eingabe einer negativen Dezimalzahl (–1 bis –32768) wird die Zahl in hexadezimaler Schreibweise sowie als Dezimalzahl (über 32767) ausgegeben.

" ":
Nach der Eingabe einer hexadezimalen Zahl (1 bis 4 Zeichen lang) wird diese ins Dezimalsystem umgerechnet und ausgegeben. Zahlen über 32767 werden zusätzlich als "Zweier-Komplement" ausgegeben.
Bei der Ausgabe der negativen Dezimalzahlen kommt es beim EX-Basic zu Fehlern, da, anders als beim Minimem, die Dezimalzahlen auf -null- runtergerechnet werden. Die Funktionen erschienen nicht wichtig genug, um die Änderungen im Listing aufzunehmen. Mittels Zwischenvariablen kann dies einfach geändert werden.

"C":
Verzweigt zu den Cassettenrekorderroutinen.
Es kann gewählt werden, ob eine Quelldatei eingespielt wird oder auf Cassette gesichert wird.
Vor jedem Start eines neuen Maschinensprache-Programmes sollte die Quelldatei auf Cassette überspielt werden. „Hängt" sich nämlich das Programm auf, so muß fast immer der Computer ausgeschaltet werden. In diesem Fall würde die erstellte Quelldatei verloren gehen.
Trotz der langsamen Cassettenroutine ist dieses Abspeichern immer noch schneller, als die Neueingabe über die Tastatur.

"Z":
Dieses Unterprogramm stattet die Datensätze mit neuen Zeilennummern aus.
Die Zeilen beginnen bei 100 und steigen je Zeile um 5 (wie RES 100,5 des TI-Basic).
Andere Zeichen als die o.a. führen zu Fehlermeldungen. Sollen Datensätze aus der Quelldatei gelöscht werden, so muß lediglich die Zeilennummer eingegeben werden. Änderungen der Sätze sind nur durch Neueingabe möglich. Wird die Funktion "N" irrtümlich angewählt, so sind die eingegebenen Sätze nicht verloren. Im Direktmodus muß dann nach "Break" in die Variable -S- die Zahl der Sätze eingegeben werden. Mit CON wird der Assembler dann wieder gestartet.
Folgende Pseudo-Instruktionen bzw. Direktiven werden vom Assembler interpretiert, wenn sie im Operationsfeld stehen:

Allgemeines:
Bei der Beschreibung wird das Format an Hand einer Beispielszeile gegeben. Die ersten drei Zeichen stellen immer die Zeilennummer dar. Steht ein Label im Kennzeichenfeld, so ist die Benutzung von diesen wahlfrei (auf Besonderheiten wird hingewiesen).
a = Beispiel
b = Beschreibung

1. "NOP"
a) 100 LB NOP
b) Pseudoinstruktion; identisch mit JMP +2. Das Operandenfeld wird völlig ignoriert.

2. "RT "
a) 520 B1 RT
b) Pseudoinstruktion; identisch mit B *R11 Das Operandenfeld wird völlig ignoriert.

3. "DEF "
a) 670 LB DEF WORT. @Operand
b) WORT kann 1 – 6 Zeichen lang sein (wird immer intern mit Leerzeichen (ASCII 32) auf 6 Zeichen aufgefüllt). WORT wird als Programmname eines Maschinensprache-Programms in die DEF-Table eingetragen. Der Operand nach dem "at"-Zeichen kann eine Hexadezimal- oder eine Dezimalzahl oder ein Symbol sein. Es muß die Adresse enthalten, an der WORT startet. Die Adresse wird in die DEF-Table eingetragen. Das Unterprogramm "DEF" verändert Zeiger auf die DEF-Table im RAM. Treten nach "DEF" Fehler im Assemblerprogramm auf, so springt der Assembler zur Eingabezeile zurück. Vor einem neuen

Assemblerdurchlauf muß der Zeiger im Direktmodus zurückgesetzt werden. Es ist daher sinnvoll, "DEF" erst unmittelbar vor der "END"-Zeile zu benutzen. Trotzdem sollte der Wert erst gesichert werden. Beim Minimem steht der Wert an Adresse >701E (28702), beim EX-Basic in >2004 (8196). Mit dem Unterprogramm "S" ist der Wert leicht festzustellen.
Wird ein Label benutzt, so wird dieser der momentanen Adresse zugewiesen.
Im Regelfall sollte kein Label benutzt werden.

4. "EQU" nn

a) 130 WT EQU >7D00
b) Einem Label wird ein bestimmter Wert (nn) zugewiesen, der zwischen 0 und 65535 liegen muß. Ein Label muß benutzt werden, da sonst ein Fehler auftritt.

5. "BSS" nn

a) 130 BF BSS 32
b) Es werden nn Bytes Speicherplatz reserviert. Sie haben keinen vordefinierten Wert.
Ist der Wert nn ungerade, so wird ein Byte dazuaddiert, damit die nächste Anweisung an einer geraden Adresse beginnt.
Ein Label muß benutzt werden, da sonst ein Fehler auftritt. Die Adresse des ersten Byte wird dem Label zugeordnet.
Diese Direktive wird benutzt, um Stringbuffer oder Platz für ein Workspace zu reservieren.

6. "DATA" nn

a) 126 VW DATA >2000
b) Die Direktive weist dem nächsten Wort den Wert nn zu. Der Wert muß zwischen 0 und 65535 liegen. Es können auch Symbole als Operand benutzt werden.
Es ist immer nur ein Wert pro DATA-Direktive möglich.

7. "TEXT"

a) 980 AS TEXT ASCII-Zeichen
b) Diese Direktive speichert ab der augenblicklichen Adresse einen Text ab. Alle ASCII-Zeichen, die über die Tastatur eingegeben werden können sind erlaubt.
Der Text beginnt ab dem 13. Zeichen des Datensatzes. Er muß nicht in Anführungsstriche gesetzt werden.
28 Zeichen können pro Text-Direktive eingegeben werden.
Komma wird als FCTN -C- (ASCII-Zeichen 96) eingegeben.
Abschließende Leerzeichen sind nicht möglich, da diese bei der Eingabe vom TI-Basic gestrichen werden.
Die beiden Einschränkungen gelten nicht, wenn der gesamte Datensatz in Anführungsstrichen eingegeben wird.
Auf das Format ist hierbei zu achten. Es dürfen nie 5 TEXT-Direktiven mit 28 Zeichen Text hintereinander in der Quelldatei stehen, da dies die Cassettenroutinen nicht erlauben. Es käme zu einer TI-Basic-Fehlermeldung mit Abbruch des Assemblers.
Ein Programm darf auch nicht viele TEXT-Direktiven mit längerem Text beinhalten. Es könnte zu einer TI-Basic-Fehlermeldung (Memory Full) führen.
Wird die Anzahl der Zeichen im Operandenfeld auf 16 beschränkt, so treten beide Möglichkeiten nicht auf. Zudem bleibt das Listing der Quelldatei übersichtlich.
Wird eine ungerade Anzahl von Text-Zeichen 32 <Space> an, damit die nächste Anweisung an einer geraden Adresse beginnt.
Die Adresse, an der das erste Zeichen steht, wird einem benutzten Label zugewiesen.

8. "AORG"

a) 100 AORG >7118
b) Die AORG-Anweisung muß immer im Operationsfeld des ersten Datensatzes stehen.
Ein benutzter Label wird ignoriert.
Die Benutzung des Operanden ist wahlfrei. Wird kein Operand benutzt, so liest die Anweisung die nächste freie Adresse (FFAM) aus dem Speicher und ab dieser Adresse wird das Maschinensprache-Programm ins RAM abgespeichert.
Die Werte werden beim Minimem aus der Adresse >701C (28700 dez.) und aus dem EX-Basic aus >2002 (8294 dez.) ausgelesen.
Wird ein Operand benutzt, so wird dieser berechnet. Ab dieser Adresse erfolgt die Abspeicherung.
Symbole sind nicht erlaubt.
Der Assembler führt nur eine geringe Kontrolle durch, ob die selbstgewählte Adresse sinnvoll ist (ob vorhandene Programme überschrieben werden, dies wird nicht überprüft).
Beim Minimem ist der Bereich von > 7000 – >7FFF, beim EX-Basic der von> 2474 – >3FFF zulässig.
Werden die Basic-Zeilen 680 und 1930 des Assemblers gestrichen, so ist jeder Speicherbereich zulässig.

9. "END"

a) 999 END #
b) Die "END"-Direktive muß immer im letzten Datensatz der Quelldatei stehen.
Wird ein Label benutzt, so wird dieser ignoriert.

Folgende Adressen sind durch Erwähnung des Namens mit vorangehenden "at"-Zeichens erreichbar:

Name:	Adresse (hexadezimal) a) Mini-Memory	b) EX-Basic
UTLTA8	7020	----
PAD	8300	8300
GPLWS	83E0	83E0
SOUND	8400	8400
VDPRD	8800	8800
VDPSTA	8802	8802
VDPWD	8C00	8C00
VDPWA	8C02	8C02
SPCHRD	9000	9000
SPCHWT	9400	9400
GRMRD	9800	9800
GRMRA	9802	9802
GRMWD	9C00	9C00
GRMWA	9C02	9C02
SCAN	000E	000E
XMLINK	601C	2018
KSCAN	6020	201C
VS8W	6024	2020
VM8W	6028	2024
VS8R	602C	2028
VM8R	6030	202C
VWTR	6034	2030
DSRLNK	6038	----
LOADER	603C	----
GPLLNK	6018	----
NUMASG	6040	2008
NUMREF	6044	200C
STRASG	6048	2010
STRREF	604C	2014
ERR	6050	2034
FAC	----	834A
FADD	----	0D80
FSU8	----	0D7C
FMUL	----	0E88
FDIV	----	0FF4
SADD	----	0D84
SSU8	----	0D74
SMUL	----	0E8C
SDIV	----	0FF8
CSN	----	11AE
CFI	----	1288
FCOMP	----	0D3A
NEXT	----	0070
ASCII	----	8375
STATUS	----	837C
ARG	----	835C

Änderungen beim Minimem sind nicht möglich, da hier die Adressen aus Platzgründen aus dem ROM ausgelesen werden.
Änderungen beim EX-Basic sind möglich, da Namen und dazugehörende Adressen im Basic-Programm definiert werden.
Beispiele:

```
100 lb BLWP @XMLINK
110    MOV  @FAC.
            @ARG
120 A  MOVB @ASCII.
            R1
130    CB   @STATUS.
            R7
```

```
100 REM ASSEMBLER
110 REM (c) UWE WIEMER '85
120 DIM S$(150),O(3),L(30),E
(40),E$(40)
130 D$="XOP NOP RT  DEF EQU
BSS DATATEXT"
140 M$="A   AB  ABS AI  ANDI
B   BL  BLWPC   CB  CI  CLR
COC CZC DEC DECTDIV INC INCT
INV JEQ JGT JH  JHE JL  JLE"
150 M$=M$&" JLT JMP JNC JNE
JNO JOC JOP LDCRLI  LIMILWPI
MOV MOVBMPY NEG ORI RTWPS
SB  SBO SBZ SETOSLA SOC "
160 M$=M$&"SOCBSRA SRC SRL S
TCRSTSTSTWPSWPBSZC SZCBTB  X
   XOR "
170 C$="A001B0010746022B0248
0446068604060800190010288040C6
2003240306060646 3C03058605C6
0546130215021B0214021A02"
180 C$=C$&"12021102100217021
6021902180021C0230040208030B0
2E8C001D0013803050602680387 6
00170011D021E0207060A05E001"
190 C$=C$&"F00108050B05090 53
40402C802AB06C6400150011F020
4862803"
200 S=0
210 CALL CLEAR
220 PRINT :"#__ __ #_'_ OP"
230 INPUT "":T$
240 D=LEN(T$)
250 IF (D>37)+(D=0)THEN 560
260 ON POS("NHAMSLR>.-CZ",T$
,1)+1 GOTO 270,200,210,580,7
30,810,970,1210,1250,1310,13
70,1420,1590
270 V$=SEG$(T$,1,3)
280 GOSUB 3960
290 IF ZW<100 THEN 3560
300 IF D=3 THEN 380
310 IF S=150 THEN 3730
320 T$=T$&"    "
330 IF (SEG$(T$,4,1)<>" ")+(
SEG$(T$,7,1)<>" ")+(SEG$(T$,
8,1)=" ")+(SEG$(T$,12,1)<>"
")THEN 560
340 IF D>12 THEN 360
350 D=11
360 T$=SEG$(T$,1,D)
370 IF (V$>SEG$(S$(S),1,3))+
(S=0)THEN 530
380 GOSUB 4020
390 IF (D=3)+(V$<>SEG$(S$(I)
,1,3))=-2 THEN 550
400 IF V$<SEG$(S$(I),1,3)THE
N 470
410 IF D<>3 THEN 510
420 FOR J=I TO S-1
430 S$(J)=S$(J+1)
440 NEXT J
450 S=S-1
460 GOTO 550
470 FOR J=S TO I STEP -1
480 S$(J+1)=S$(J)
490 NEXT J
500 S=S+1
510 S$(I)=T$
520 GOTO 550
530 S=S+1
540 S$(S)=T$
550 GOTO 230
560 PRINT :"EINGABE?"
570 GOTO 4290
580 T$=S$(1)
590 IF (SEG$(S$(S),8,4)<>"EN
D ")+(SEG$(T$,8,4)<>"AORG")T
HEN 4270
600 IF LEN(T$)<12 THEN 650
610 O$=SEG$(T$,13,6)
620 GOSUB 4070
630 A=D-(D/2<>INT(D/2))
640 GOTO 670
650 CALL PEEK(28700,I,J)
660 A=I*256+J
670 PRINT : : :"AORG";A: : :
680 IF A<28672 THEN 950
690 M=1
700 Z=1
710 L$=""
720 GOTO 1630
730 PRINT :"SAETZE:";S,"  ="
;150-S;"FREI"
740 PRINT "LABEL :";Z-1-(Z=0
): :
750 FOR I=1 TO Z-1
760 IF I>9 THEN 780
770 PRINT " ";
780 PRINT ;STR$(I);". ";SEG$
(L$,I*2+1,2);" @";STR$(L(I))
;
790 NEXT I
800 GOTO 220
810 INPUT "ADR.:":O$
820 GOSUB 4070
830 K=D
840 IF (K<-32768)+(K>65536)T
HEN 950
850 IF K<32767 THEN 870
860 K=K-65536
870 CALL PEEK(K,D,J)
880 D=D*256+J
890 GOSUB 3840
900 PRINT K;TAB(10);"=> ";H$
910 K=K+2
920 CALL KEY(0,R,T)
930 IF R=13 THEN 220
```

FEHLERMELDUNGEN:

Die Fehlermeldungen sind in der Regel eindeutig, da zu dem Zeitpunkt, zu dem sie ausgegeben werden, auch der Fehler auf dem Bildschirm zu sehen ist.
In der Regel befindet sich der Fehler in der letzten ausgegebenen Zeile.

Einige Besonderheiten:

"Überlauf" –
während der Eingabe der Datensätze: es sind mehr als 150 Sätze vorhanden
– während des Assembliervorganges: es sind mehr als 30 Label oder 40 Symbole vorhanden

"?" mit Datensatz, Adresse und Label
– es wird ein Symbol benutzt und es gibt keinen entsprechenden Label
– das letzte Zeichen eines Labels und das erste Zeichen des nächsten Labels ergeben zusammen ein benutztes Symbol

"Eingabe"
– wird ein Datensatz eingegeben, stimmt das Format nicht bzw. das Operationsfeld ist leer
– unzulässige Eingaben im Editor

"Operand/Znr?"
– wird ein Datensatz eingegeben, so stimmt die Zeilennummer nicht
– im Editor: falsche Dezimal- oder Hexadezimalzahl
– während des Assemblerdurchlaufes: im Operandenfeld steht ein ungültiger Operand

"Label"
– das erste Zeichen eines Labels ist nicht alphabetisch
tritt dies bei einem Symbol auf, so wird die Meldung "Operand/Znr?" gegeben.

"Adr.?"
– es wird an eine unzulässige Adresse geladen
– falsche Adresse (>65535 oder<-32768)

Die END-Anweisung berechnet die erste freie Adresse im RAM. Dieser Wert wird dem Label "EX" zugewiesen. Auf diese Adresse kann im Programm im Operandenfeld durch das Symbol ("EX") bezug genommen werden.
Dieser Label "EX" sollte in der Quelldatei als Label nicht noch einmal verwandt werden, da es mit Sicherheit zu Fehlberechnungen kommt und ein Maschinensprache-Programm mit Sicherheit abstürzt.
Die Benutzung des "File"-Zeichens ist wahlfrei. Wird es benutzt, so wird nach fehlerfreiem Assemblerdurchlauf der Zeiger FFAM (First Free Adress in Memory) nicht geändert.
Dies ist zum Austesten des Assemblers gedacht. Es ist aber auch wichtig, wenn einzelne Worte eines Maschinensprache-Programms geändert werden sollen.
Beispiel:
100 AORG Adresse
110 DATA WERT
120 END #
Auf FFAM wird nicht eingewirkt.
Wird das "File"-Zeichen weggelassen, so wird FFAM geändert (gleich zu "EX" gesetzt).

10." ! "

a) 120 ! Kommentar
b) Benutzte Label werden ignoriert.
Da kein Kommentarfeld in den Datensätzen vorgesehen ist, wurde die Möglichkeit eingeräumt, Kommentare in eigenen Datensätzen ins Assemblerlisting einzufügen.
Das Program wird lesbarer.
Es gelten die selben Einschränkungen hinsichtlich der Länge wie bei der Anweisung "TEXT".

Uwe Wiemer



```
940 IF T<>Ø THEN 920 ELSE 85
Ø
950 PRINT :"ADR.?"
960 GOTO 429Ø
970 INPUT "ZNR.: ":T$
980 IF (T$="")+(LEN(T$)>3)TH
EN 56Ø
990 IF T$="A" THEN 1Ø8Ø
1000 IF T$<>"L" THEN 1Ø4Ø
1010 IF S<14 THEN 1Ø8Ø
1020 D=S-14
1030 GOTO 1Ø8Ø
1040 V$=SEG$(T$,1,3)
1050 GOSUB 397Ø
1060 GOSUB 4Ø2Ø
1070 D=I
1080 K=Ø
1090 FOR I=D TO S
1100 K=K+1
1110 PRINT S$(I)
1120 IF K<21 THEN 119Ø
1130 CALL KEY(Ø,R,T)
1140 IF T=Ø THEN 113Ø
1150 IF R=13 THEN 22Ø
1160 K=2Ø
1170 IF R=32 THEN 119Ø
1180 K=Ø
1190 NEXT I
1200 GOTO 23Ø
1210 INPUT "NAME: ":Q$
1220 IF Q$="" THEN 56Ø
1230 CALL LINK(Q$)
1240 GOTO 22Ø
1250 INPUT "HEX:>":H$
1260 GOSUB 389Ø
1270 PRINT D
1280 IF D<32768 THEN 13ØØ
1290 PRINT D-65536
1300 GOTO 22Ø
1310 INPUT "DEZ: ":V$
1320 GOSUB 396Ø
1330 D=ZW
1340 GOSUB 383Ø
1350 PRINT ">";H$
1360 GOTO 128Ø
1370 INPUT "DEZ: -":V$
1380 GOSUB 41ØØ
1390 GOSUB 383Ø
1400 PRINT D:" >";H$
1410 GOTO 22Ø
1420 INPUT "1=SAVE 2=OLD (CS
1): ":V$
1430 IF V$="2" THEN 152Ø
1440 IF V$<>"1" THEN 56Ø
1450 OPEN #1:"CS1",INTERNAL,
OUTPUT,FIXED 192
1460 FOR I=1 TO 15Ø STEP 5
1470 PRINT #1:S,S$(I),S$(I+1
),S$(I+2),S$(I+3),S$(I+4)
1480 IF S$(I+4)="" THEN 15ØØ
1490 NEXT I
1500 CLOSE #1
1510 GOTO 158Ø
1520 OPEN #1:"CS1",INTERNAL,
INPUT ,FIXED 192
1530 FOR I=1 TO 15Ø STEP 5
1540 INPUT #1:S,S$(I),S$(I+1
),S$(I+2),S$(I+3),S$(I+4)
1550 IF S$(I+4)="" THEN 157Ø
1560 NEXT I
1570 CLOSE #1
1580 GOTO 22Ø
1590 FOR I=1 TO S
1600 S$(I)=STR$(95+I*5)&SEG$
(S$(I),4,34)
1610 NEXT I
1620 GOTO 22Ø
1630 FOR J=2 TO S-1
1640 T$=S$(J)
1650 PRINT T$
1660 IF SEG$(T$,8,1)="!" THE
N 198Ø
1670 CALL KEY(Ø,R,T)
1680 IF R=13 THEN 22Ø
1690 B=Ø
1700 IF SEG$(T$,5,2)="  " TH
EN 179Ø
1710 IF (SEG$(T$,5,1)>="A")+
(SEG$(T$,5,1)<="Z")=-2 THEN
174Ø
1720 PRINT :"LABEL?"
1730 GOTO 429Ø
1740 IF Z>3Ø THEN 373Ø
1750 L$=L$&SEG$(T$,5,2)
1760 L(Z)=A
1770 B=1
1780 Z=Z+1
1790 W=1
1800 B$=SEG$(T$,8,4)
1810 P=POS(M$,B$,1)
1820 IF P THEN 187Ø
1830 ON INT((POS(D$,B$,1)+3)
/4)+1 GOTO 428Ø,184Ø,273Ø,27
5Ø,277Ø,294Ø,299Ø,3Ø5Ø,3Ø9Ø
1840 K=9
1850 C=11264
1860 GOTO 191Ø
1870 CALL CHAR(143,SEG$(C$,P
,3))
1880 CALL PEEKV(1912,Y,H)
1890 C=Y*256+H
1900 K=VAL(SEG$(C$,P+3,1))
1910 ON K GOSUB 223Ø,229Ø,23
9Ø,239Ø,247Ø,255Ø,233Ø,259Ø,
239Ø
1920 FOR I=1 TO W
1930 IF A>32768 THEN 95Ø
1940 R=INT(O(I)/256)
1950 CALL LOAD(A,R,O(I)-R*25
6)
1960 A=A+2
1970 NEXT I
1980 NEXT J
1990 PRINT S$(J):
2000 L$="EX"&L$
2010 L(Ø)=A
2020 FOR J=1 TO M-1
2030 D=(POS(L$,SEG$(E$(J),1,
2),1)+1)/2
2040 IF D<>INT(D)THEN 2Ø9Ø
2050 D=L(D-1)
2060 IF E(J)>Ø THEN 213Ø
2070 K=(D+E(J)-2)/2
2080 IF (K>127)+(K<-128)=Ø T
HEN 211Ø
2090 PRINT :"?":S$(VAL(SEG$(
E$(J),3,3))):"ADR.:";ABS(E(J
));"LABEL: ";SEG$(E$(J),1,2)
2100 GOTO 429Ø
2110 CALL LOAD(-E(J)+1,K)
2120 GOTO 215Ø
2130 I=INT(D/256)
2140 CALL LOAD(E(J),I,D-256*
I)
2150 NEXT J
2160 PRINT :"ENDADR:";A;
2170 IF SEG$(S$(S),13,1)="#"
 THEN 221Ø
2180 T=INT(A/256)
2190 CALL LOAD(287ØØ,T,A-256
*T)
2200 PRINT ;"  = FFAM!";
2210 PRINT ;"(EX)";
2220 GOTO 73Ø
2230 GOSUB 324Ø
2240 O(1)=C+16*T+R
2250 O$=SEG$(T$,G+1,12)
2260 GOSUB 327Ø
2270 O(1)=O(1)+1Ø24*T+64*R
2280 RETURN
2290 O$=SEG$(T$,13,3)&" "
2300 IF SEG$(O$,1,1)<"A" THE
N 235Ø
2310 GOSUB 423Ø
2320 E(M-1)=-E(M-1)
2330 D=Ø
2340 GOTO 237Ø
2350 GOSUB 415Ø
2360 IF D>255 THEN 356Ø
2370 O(W)=C+D
2380 RETURN
2390 GOSUB 324Ø
2400 O(1)=C+16*T+R
2410 V$=SEG$(T$,G+1-(K=3),2)
2420 IF (K=3)+(SEG$(T$,G+1,1
)<>"R")=-2 THEN 356Ø
2430 GOSUB 396Ø
2440 IF ZW>15 THEN 356Ø
2450 O(1)=O(1)+64*ZW
2460 RETURN
2470 IF SEG$(T$,13,1)<>"R" T
HEN 356Ø
2480 GOSUB 324Ø
2490 O(1)=C+R
2500 V$=SEG$(T$,G+1,2)
2510 GOSUB 396Ø
2520 IF ZW>15 THEN 356Ø
2530 O(1)=O(1)+16*ZW
2540 RETURN
2550 G=25
2560 GOSUB 326Ø
2570 O(1)=C+16*T+R
2580 RETURN
2590 IF POS("LIMILWPI",B$,1)
=Ø THEN 263Ø
2600 O(1)=C
2610 G=12
2620 GOTO 267Ø
2630 IF SEG$(T$,13,1)<>"R" T
HEN 356Ø
2640 GOSUB 324Ø
2650 O(1)=C+R
2660 IF SEG$(T$,8,1)="S" THE
N 272Ø
2670 O$=SEG$(T$,G+1,6)
2680 W=W+1
2690 GOSUB 4Ø7Ø
2700 IF (B$="LIMI")+(D>2)=-2
 THEN 356Ø
2710 O(2)=D
2720 RETURN
2730 O(1)=4Ø96
2740 GOTO 192Ø
2750 O(1)=1115
2760 GOTO 192Ø
2770 G=POS(T$,".",14)
2780 IF (SEG$(T$,G+1,1)<>"@"
)THEN 356Ø
2790 O$=SEG$(T$,G+2,6)
2800 Q$=SEG$(T$,13,G-13)&"
"
2810 Q$=SEG$(Q$,1,6)
2820 CALL PEEK(287Ø2,R,T)
2830 DE=A
2840 A=(R*256+T)-8
2850 R=6
2860 GOSUB 315Ø
2870 GOSUB 4Ø7Ø
2880 R=INT(D/256)
2890 CALL LOAD(A,R,D-R*256)
2900 Z1=INT((A-6)/256)
2910 CALL LOAD(287Ø2,Z1,A-6-
256*Z1)
2920 A=DE
2930 GOTO 198Ø
2940 IF B=Ø THEN 172Ø
```

```
2950 O$=SEG$(T$,13,6)
2960 GOSUB 4070
2970 L(Z-1)=D
2980 GOTO 1980
2990 IF B=0 THEN 1720
3000 V$=SEG$(T$,13,5)
3010 GOSUB 3960
3020 L(Z-1)=A
3030 A=A+ZW-(INT(ZW/2)<>ZW/2
)
3040 GOTO 1980
3050 O$=SEG$(T$,13,6)
3060 GOSUB 4070
3070 O(1)=D
3080 GOTO 1920
3090 K=LEN(T$)
3100 IF K<13 THEN 3560
3110 Q$=SEG$(T$,13,K-12)
3120 R=LEN(Q$)
3130 R=R-(INT(R/2)<>R/2)
3140 Q$=Q$&" "
3150 FOR I=1 TO R
3160 T=ASC(SEG$(Q$,I,1))
3170 IF T<>96 THEN 3190
3180 T=44
3190 CALL LOAD(A,T)
3200 A=A+1
3210 NEXT I
3220 IF B$="TEXT" THEN 1980
3230 RETURN
3240 G=POS(T$,".",13)
3250 G=G-16*(G=0)
3260 O$=SEG$(T$,13,G-13)
3270 Z$=SEG$(O$,1,1)
3280 ON POS("R*@",Z$,1)+1 GO
TO 3560,3290,3330,3420
3290 T=0
3300 V$=SEG$(O$,2,2)
3310 GOSUB 3960
3320 GOTO 3390
3330 T=1
3340 R=POS(O$,"+",4)
3350 IF R=0 THEN 3370
3360 T=3
3370 V$=SEG$(O$,3,LEN(O$)+(R
<>0)-2)
3380 GOSUB 3960
3390 R=ZW
3400 IF R>15 THEN 3560
3410 GOTO 3820
3420 W=W+1
3430 R=0
3440 T=2
3450 GOTO 3580
3460 Q$=SEG$(O$,2,6)
3470 FOR I=28430 TO 28662 ST
EP 8
3480 FOR R1=0 TO 5
3490 CALL PEEK(I+R1,R2)
3500 IF CHR$(R2)<>SEG$(Q$,R1
+1,1)THEN 3550
3510 NEXT R1
3520 CALL PEEK(I+6,R1,R2)
3530 D=R1*256+R2
3540 GOTO 3810
3550 NEXT I
3560 PRINT :"OPERAND/ZNR?"
3570 GOTO 4290
3580 U=POS(O$,"(",3)
3590 IF U=0 THEN 3660
3600 V=POS(O$,")",6)
3610 IF V=0 THEN 3560
3620 V$=SEG$(O$,U+2,V-U-2)
3630 GOSUB 3960
3640 R=ZW
3650 IF R>15 THEN 3560
3660 U=U-(U=0)*8
3670 IF SEG$(O$,2,1)<"A" THE
N 3790
3680 O$=O$&"    "
3690 IF LEN(O$)>6 THEN 3460
3700 A$=SEG$(O$,2,2)
3710 IF SEG$(A$,1,1)>"Z" THE
N 3560
3720 IF M<40 THEN 3750
3730 PRINT :"UEBERLAUF"
3740 GOTO 4290
3750 E(M)=A+W*2-2
3760 E$(M)=A$&STR$(J)
3770 M=M+1
3780 RETURN
3790 O$=SEG$(O$,2,U-2)
3800 GOSUB 4070
3810 O(W)=D
3820 RETURN
3830 IF D>65536 THEN 3560
3840 Y=INT(D/256)
3850 CALL POKEV(1912,Y,D-Y*2
56)
3860 CALL CHARPAT(143,H$)
3870 H$=SEG$(H$,1,4)
3880 RETURN
3890 FOR I=1 TO LEN(H$)
3900 IF POS("0123456789ABCDE
F",SEG$(H$,I,1),1)=0 THEN 35
60
3910 NEXT I
3920 CALL CHAR(143,H$)
3930 CALL PEEKV(1912,Y,H)
3940 D=(Y*256+H)/2^(16-(4*LE
N(H$)))
3950 RETURN
3960 IF V$="" THEN 3560
3970 FOR I=1 TO LEN(V$)
3980 IF POS("0123456789",SEG
$(V$,I,1),1)=0 THEN 3560
3990 NEXT I
4000 ZW=VAL(V$)
4010 RETURN
4020 FOR I=1 TO S
4030 IF V$<=SEG$(S$(I),1,3)T
HEN 4060
4040 NEXT I
4050 I=I-1
4060 RETURN
4070 IF SEG$(O$,1,1)=">" THE
N 4200
4080 IF SEG$(O$,1,1)<>"-" TH
EN 4140
4090 V$=SEG$(O$,2,6)
4100 GOSUB 3960
4110 IF ZW>32768 THEN 3560
4120 D=65536-ZW
4130 GOTO 4260
4140 IF SEG$(O$,1,1)>="A" TH
EN 4230
4150 V$=O$
4160 GOSUB 3960
4170 D=ZW
4180 IF D>65535 THEN 3560
4190 GOTO 4260
4200 H$=SEG$(O$,2,4)
4210 GOSUB 3890
4220 GOTO 4260
4230 A$=SEG$(O$,1,2)
4240 IF (B$="EQU ")+(T$=S$(1
))+(SEG$(T$,1,1)="S")+(B$="L
IMI")THEN 3560
4250 GOSUB 3710
4260 RETURN
4270 PRINT :"AORG/END";
4280 PRINT :"MNEMONIC ?"
4290 CALL SOUND(190,219,0)
4300 GOTO 220
```

```
60 RF$="VDPWA VDPWD VDPRD VD
PSTAFAC   GPLWS PAD   SOUND 
SPCHRDSPCHWTGRMRD GRMRA GRMW
D GRMWA SCAN  NUMASGNUMREFST
RASGSTRREFXMLLNKKSCAN "
70 RF$=RF$&"VSBW  VM8W  VSBR
  VM8R  VWTR  ERR   FADD  FS
UB  FMUL  FDIV  SADD  SSUB  
SMUL  SDIV  CSN   CFI   FCOM
P NEXT  ASCII STATUSARG    "
80 RFA$="3584235840348163481
8336103376033536337923686437
88838912389143993639938    14
 8200 8204 8208 8212 8216 82
20 8224 8228 8232 8236 8240"
90 RFA$=RFA$&" 8244 3456 345
2 3720 4048 3460 3444 3724 4
088 4526 4792 3386  11233653
3366033628"
650 CALL PEEK(8194,I,J)
680 IF A<9460 THEN 950
1870 H$=SEG$(C$,P,3)&"0"
1880 GOSUB 3890
1890 C=D
1930 IF A>16383 THEN 950
2190 CALL LOAD(8194,T,A-256*
T)
2820 CALL PEEK(8196,R,T)
2910 CALL LOAD(8196,Z1,A-6-2
56*Z1)
3470 RFP=POS(RF$,Q$,1)
3480 IF RFP=0 THEN 3560
3490 RFP=(RFP-1)/6
3500 RFA=(RFP*5)+1
3510 D=VAL(SEG$(RFA$,RFA,5))
3520 GOTO 3810
3530 REM
3540 REM
3550 REM
3840 H$="" :: Y=4096
3845 FOR I=0 TO 3
3850 H=INT(D/Y)
3855 H$=H$&CHR$(H+48-(7*(H>9
)))
3860 D=D-Y*H
3865 Y=Y/16
3870 NEXT I
3890 D=0 :: Y=1
3895 FOR I=LEN(H$)TO 1 STEP
-1
3900 H=POS("0123456789ABCDEF
",SEG$(H$,I,1),1)
3910 IF H=0 THEN 3560
3920 D=D+(H-1)*Y
3930 Y=Y*16
3940 NEXT I
```

HARD-COPY

Hardcopy ist eigentlich ein alter Hut. Jedoch werden diejenigen unter Ihnen, die zum Beispiel einen Seikosha GP 100 A besitzen, festgestellt haben, daß die Sache mit der Hardcopy bei diesem Druckertyp etwas komplizierter ist als sonst.
Einige von Ihnen werden sicherlich schon einmal versucht haben, eine in Basic erstellte Grafik auf den Drucker zu bringen, um dann festzustellen, daß zu ihrem „Glück" die acht Pixelreihe jedes Zeichens fehlt. Grundsätzlich besteht bei allen Sieben-Nadel-Druckern dieses Problem.
Nun, dieses Programm soll es Ihnen ermöglichen, Grafiken schwarz auf weiß vor sich liegen zu haben.
Die Sache ist eigentlich ganz einfach. Man braucht ja bloß den achten Pixelpunkt der ersten Druckzeile als ersten Pixelpunkt der zweiten Pixelzeile zu nehmen und den siebten und achten Pixelpunkt der zweiten Druckzeile als ersten und zweiten Pixelpunkt der dritten Druckzeile, usw. Das macht man so lange, bis man die ersten sieben Bildschirmzeilen gedruckt hat und dann geht ab der achten Bildschirmzeile alles wieder von vorne los.
So etwas in Basic zu realisieren ist relativ einfach, aber schließlich und endlich hat nicht jeder die Zeit, bis zu dreieinhalb Stunden auf seine Grafik zu warten. Das vorliegende Assemblerprogramm erlaubt es, die gleiche Grafik in ca. 45 Sekunden zu erhalten.
Das Programm ist voll kommentiert und dürfte beim Abtippen keine Schwierigkeiten bereiten.

Hinweise zum Laden des Programms:

Editor/Assembler-Version:
CALL INIT
CALL LOAD
("DSK1.§HARDCOPY")

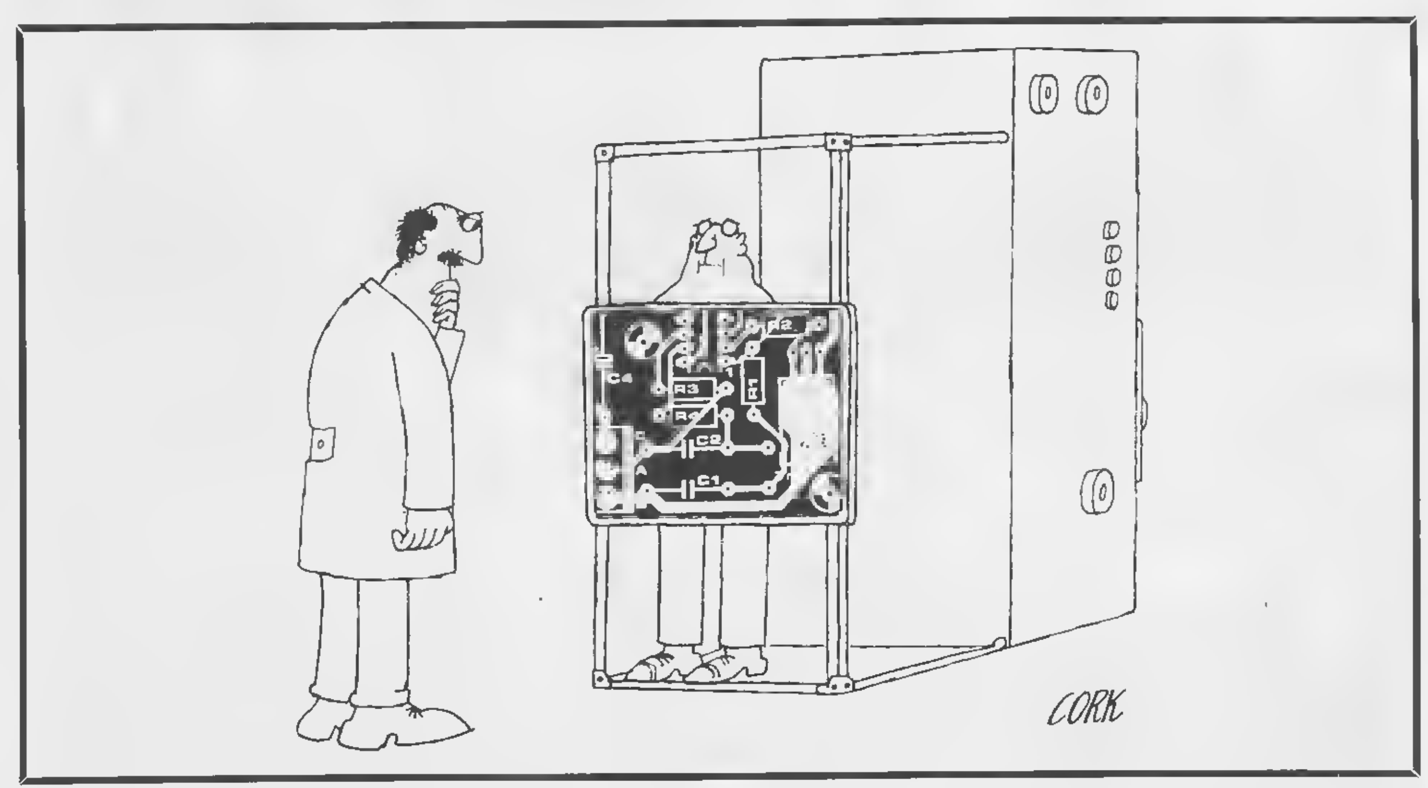
Minimemory-Version:
CALL INIT
CALL LOAD
("DSK1.§HARDCOPY2")

Minimemory-Pokeliste:
OLD CS1
RUN
oder
OLD DSK1.HARDCOPY
RUN

Das Programm wird dann mit CALL LINK ("HCOPY") aufgerufen.

Wenn Sie übrigens das Expanded-Grafik-Basic-Paket von Apesoft besitzen, so können Sie durch Veränderung des folgenden Punktes auch von diesen Grafiken Hardcopys anfertigen.

BASIS DATA >0000
in
BASIS DATA >3000
ändern.
Die Exekution erfolgt ohne jegliche Probleme.
Wie Sie bemerkt haben werden, habe ich keine Hinweise für eine Anpassung des Programms an das Extended-Basic-Modul gegeben. Dies legt daran, daß die DSRLNK- und die GPLLNK-Routine in Extended-Basic nicht vorhanden ist. Wer dennoch auch dort nicht auf eine Hardcopy verzichten möchte, den verweise ich auf dieses Sonderheft. Dort steht eine DSRLNK- und eine GPLLNK-Ersatzroutine für das Extended-Basic-Modul. Bitte versuchen Sie selber, diese Routinen in das Programm einzubauen.
Und nun viel Spaß beim Abtippen. *Bernd Bertling*



```
*      UNL
* HARDCOPY
* ========
*
* (c) by Bernd Bertling, Neu-Crengeldanzstra~e 2
*                        4600 Dortmund 72
*
* Programm f}r Bildschirm-Hardcopy auf einem Seikosha GP 100 A oder anderen
* Druckern mit nur sieben Nadeln.
*
* Notwendige Ger{tekonfiguration:
* ------------------------------
* 1.Editor/Assembler Paket zur Programmerstellung
* 2.Speichererweiterung
* 3.Editor/Assembler- oder Minimemory-Modul
* 4.Diskettensystem mit mind. 1 Laufwerk
* 5.RS232 Schnittstelle
* 6.Seikosha GP 100 A
*
* Programmnamen:  DSK1.@HARDCOPY  --> Name f}r Objekt-Code
* -------------   DSK1.@HARDCOPYS --> Name f}r Source-Code
*
* Aufruf: CALL LINK("HCOPY")
* -------
*
* Programmidentifizierung
*
       IDT  'HARDCOPY'
*
* Die untengenannten Equates m}~en bei Erstellung eines
* Objektcodes f}r das Editor/Assembler-Modul durch folgende
* Refferenzen ersetzt werden.
*
* Refferenzen f}r Editor/Assembler
*
*      REF  DSRLNK,GPLLNK,VSBW,VMBW,VSBR,VMBR
*
* Programmdefinition
*
       DEF  HCOPY
*
* Minimemory-Equates
*
DSRLNK EQU  >6038    Verbindung zur DSR-Routine
GPLLNK EQU  >6018    Verbindung zu GROM-Routinen
VSBW   EQU  >6024    Ein Byte aus VDP-RAM lesen
VMBW   EQU  >6028    Mehrere Bytes aus VDP-RAM lesen
VSBR   EQU  >602C    Ein Byte in's VDP-RAM schreiben
VMBR   EQU  >6030    Mehrere Bytes in's VDP-RAM schreiben
*
* sonstige Equates
*
STATUS EQU  >837C    Statusregister
PNTR   EQU  >8356    DSR Namenszeiger
POINT  EQU  >831C    Pab-Pointer
GPLWS  EQU  >83E0    GPL-Arbeitsbereich
*
* Dateidaten f}r den PAB
*
PAB    DATA 800      Auf >0320 im VDP-RAM liegt der Pab, da dort noch
*                    ein paar Bytes frei sind. F}r den Pab-Buffer wird
*                    mittels BLWP @GPLLNK, DATA >0038 Platz reserviert.
PAB5   BSS  2        Buffer f}r PAB-Adresse plus f}nf
PABBUF BSS  2        Buffer f}r PAB-Buffer-Adresse
PDATA  DATA >0012,>0000,>FF00,>0000,>6006
       TEXT 'PIO.CR'
D2     DATA 2
D5     DATA 5
D9     DATA 9
BASIS  DATA >0000       Basisadresse des PDT-Tables in Basic
*
* Eigene Dateidaten
*
MYDAT1 BYTE >0A,>1B,>10,>00,>68  Druckerdaten
MYDAT2 BYTE >0A,>0A,>0A,>0A,>0A  F}nfmal Zeilenvorschub
BYT0   BYTE >00
BYT128 BYTE >80         Bytekonstante
*
* Weitere Dateidaten
*
CLOSE  BYTE 1   Datei schlie~en
WRITE  BYTE 3   Zur Datei schreiben
       EVEN
*
* Der folgende Datenblock enth{lt alle Daten f}r die Auswertung
* der 24 Bildschirmzeilen, welche 28 Druckzeilen entsprechen.
*
* Der Aufbau ist folgenderma~en (Beispiel zweite Druckzeile)
*
* VDP-Adresse f}r erste Spalte und erste Zeile; VDP-Adresse f}r
* erste Spalte und zweite Zeile; Adresse der Variablen, die bearbeitet
* werden soll; Erster Schleifenz{hler f}r 'ZEILE'; Zweiter Schleifen-
* z{hler f}r 'ZEILE'
*
ZLDATA DATA >0000,>0000,STRIN2,>0007
       DATA >0000,>0020,STRIN1+7,>0106
       DATA >0020,>0040,STRIN1+6,>0205
       DATA >0040,>0060,STRIN1+5,>0304
       DATA >0060,>0080,STRIN1+4,>0403
       DATA >0080,>00A0,STRIN1+3,>0502
       DATA >00A0,>00C0,STRIN1+2,>0601
       DATA >00C0,>00C0,STRIN1+1,>0700
       DATA >00E0,>00E0,STRIN2,>0007
       DATA >00E0,>0100,STRIN1+7,>0106
       DATA >0100,>0120,STRIN1+6,>0205
       DATA >0120,>0140,STRIN1+5,>0304
       DATA >0140,>0160,STRIN1+4,>0403
       DATA >0160,>0180,STRIN1+3,>0502
       DATA >0180,>01A0,STRIN1+2,>0601
       DATA >01A0,>01A0,STRIN1+1,>0700
       DATA >01C0,>01C0,STRIN2,>0007
       DATA >01C0,>01E0,STRIN1+7,>0106
       DATA >01E0,>0200,STRIN1+6,>0205
       DATA >0200,>0220,STRIN1+5,>0304
       DATA >0220,>0240,STRIN1+4,>0403
       DATA >0240,>0260,STRIN1+3,>0502
       DATA >0260,>0280,STRIN1+2,>0601
       DATA >0280,>0280,STRIN1+1,>0700
```

```
       DATA >02A0,>02A0,STRIN2,>0007
       DATA >02A0,>02C0,STRIN1+7,>0106
       DATA >02C0,>02E0,STRIN1+6,>0205
       DATA >02E0,>02E0,STRIN1+5,>0300
*
* Returnlevel f}r verschiedene Unterprogramme
*
BACK1  BSS  2
BACK2  BSS  2
*
* Arbeitsregister
*
MYREGI BSS  32  1.Registersatz
MYREG2 BSS  32  2.Registersatz
REOI   BSS  2   Hilfsregister
REOIHB EQU  REGI
REGILB EQU  REOI+1
*
* Hilfsvariablen
*
HELP1  BSS  2
HELP2  BSS  2
HELP3  BSS  2
MATRIX BSS  56  Bin{rmatrix
STRING BSS  B   Variable f}r 8 Byte langen Seikosha-String
*
* Hilfsvariablen f}r 8 Byte lange TI-Strings
*
STRIN1 BSS  B
STRIN2 BSS  8
*
*
HCOPY  LWPI MYREGI             1.Satz von Arbeitsregistern laden
*
       LI   R0,>00FF           255 Bytes sollen f}r den Pab-Buffer
       MOV  R0,@>830C          reserviert werden.
*
       MOVB @BYT0,@STATUS      GPL-Statusbyte l|schen
       LWPI GPLWS              GPL-Arbeitsregister laden
       BLWP @GPLLNK            Platz im VDP-RAM reservieren
       DATA >003B
*
       LWPI MYREOI             1.Satz von Arbeitsregistern laden
       MOV  @>B31A,@PABBUF     Die erste freie Adresse in die Variable
*
       MOV  @PAB,R0            Adresse des Pab nach R0
       MOV  R0,@POINT          Adresse des Pab in den Pab-Pointer schieben
       A    @D5,R0             F}nf hinzuaddieren
       MOV  R0,@PAB5           Adresse in den Buffer schieben
*
       MOV  @PAB,R0            PAB-Adresse nach R0
       LI   R1,PDATA           Dateidaten-Adresse nach R1
       LI   R2,16              16 Bytes sind zu }bertragen
       BLWP @VMBW              Bytes in's VDP-RAM schreiben
*
       MOV  @PAB,R0            An diese Adresse mu~ der Pab-Buffer
       A    @D2,R0
       LI   R1,PABBUF          R1 mit der Adresse der Variablen laden
       LI   R2,2               2 Bytes sollen }bertragen werden
```

```
       BLWP @VMBW              Bytes in's VDP-RAM bringen
*
       MOV  @PAB,R6            R6 auf Dateinamen richten
       A    @D9,R6
       BL   @INOUT             Datei er|ffnen
*
       MOVB @WRITE,R1          Ausgabe-Befehl in's PAB bringen
       MOV  @PAB,R0
       BLWP @VSBW
       BL   @INOUT             Datei auf Ausgabe {ndern
*
       LI   R1,>0100           Ein Byte zum Drucker schicken
       MOV  @PAB5,R0
       BLWP @VSBW
*
       MOV  @PABBUF,R0
       LI   R1,>0B00           Grafikcode in den PAB-Buffer schreiben
       BLWP @VSBW
       BL   @INOUT             Drucker auf Grafik umstellen
*
       LI   R1,>0500           F}nf Bytes zum Drucker schicken
       MOV  @PAB5,R0
       BLWP @VSBW
*
       MOV  @PABBUF,R0         Den Ausdruck erst ab Position 6B beginnen
       LI   R1,MYDAT1
       LI   R2,5
       BLWP @VMBW
       BL   @INOUT             Daten zum Drucker schicken
*
       LI   R10,28             Druckzeilen-Z{hler
       LI   R8,ZLDATA          Adresse der Zeilenbearbeitungsdaten
*
NEXT1  LI   R15,0              R15=0
       MOV  *RB,R13            1.VDP-Adresse nach R13
       INCT R8                 Adresse=Adresse+2
       MOV  *R8,R14            2.VDP-Adresse nach R14
       INCT R8                 Adresse=Adresse+2
       MOV  *RB,@HELP1         Derzeitige Variablen-Adresse nach HELP1
       INCT R8                 Adresse=Adresse+2
       MOVB *R8,@HELP2         1.Schleifenz{hler nach HELP2
       INC  R8                 Adresse=Adresse+1
       MOVB *R8,@HELP3         2.Schleifenz{hler nach HELP3
       INC  R8                 Adresse=Adresse+1
*
NEXT2  MOV  R13,R0             1.VDP-Adresse nach R0 kopieren
       INC  R13                VDP-Adresse um eins erh|hen
       BLWP @VSBR              Zeichen aus Namensliste lesen
       SRL  R1,8               Byte in Wort verwandeln
       SLA  R1,3               Zeichen mal acht
       MOV  R1,R0              Definitionslisten-Adresse nach R0 kopieren
       MOV  @BASIS,R1          P.D.T.-Adresse nach R1
       A    R1,R0              Plus Definitionsbasisadresse
       LI   R1,STRIN1          Adresse f}r ersten Acht-Byte-Buffer laden
       LI   R2,8               Acht Bytes sind zu lesen
       BLWP @VMBR              Bytes aus VDP-RAM lesen
       MOV  R14,R0             2.VDP-Adresse nach R0 kopieren
       INC  R14                VDP-Adresse um eins erh|hen
       BLWP @VSBR              Zeichen aus Namensliste lesen
```

```
       SRL  R1,8              Byte in Wort verwandeln
       SLA  R1,3              Zeichen mal acht
       MOV  R1,RØ             Definitionslisten-Adresse nach RØ kopieren
       MOV  @BASIS,R1         P.D.T-Adresse nach R1
       A    R1,RØ             Plus Definitionsbasisadresse
       LI   R1,STRIN2         Adresse f}r zweiten Acht-Byte-Buffer laden
       LI   R2,8              Acht Bytes sind zu lesen
       BLWP @VMBR             Bytes aus VDP-RAM lesen
       LWPI MYREG2            2.Satz von Arbeitsregistern laden
       CLR  R1                R1=Ø
       LI   R15,MATRIX        R15 mit der Matrixadresse laden
       MOV  @HELP1,R14        Aktuelle STRIN1- oder STRIN2-Adresse gewinnen
       LI   R13,STRIN2        R13 mit STRIN2-Adresse laden
       MOVB @HELP2,R9         Ersten Schleifenz{hlerwert laden
       MOVB R9,R9             Soll die Schleife ausgef}hrt werden ?
       JEQ  NEXT4             Nein, also n{chste Schleife
*
       SWPB R9                Ja, also Wort in Byte verwandeln
       MOVB R1,R9             MS-Byte von R9 l|schen
*
NEXT3  LI   RØ,7              RØ mit Verschiebewert laden
       MOVB *R14,R5           Byte nach R5 kopieren
       MOVB R5,R7             Byte in R7 sichern
       BL   @COPIER           Byte in Bitmatrix eintragen
       INC  R14               R14=R14+1
       DEC  R9                Schleifenz{hler um eins erniedrigen
       JNE  NEXT3             Wenn Schleife noch nicht fertig, n{chstes Byte
*
NEXT4  MOVB @HELP3,R9         Zweiten Schleifenz{hlerwert laden
       MOVB R9,R9             Soll die Schleife ausgef}hrt werden ?
       JEQ  NEXT6             Nein, also Bitmatrix auswerten
*
       SWPB R9                Ja, also Wort in Byte verwandeln
       MOVB R1,R9             MS-Byte von R9 l|schen
*
NEXT5  LI   RØ,7              RØ mit Verschiebewert laden
       MOVB *R13,R5           Byte nach R5 kopieren
       MOVB R5,R7             Byte in R7 sichern
       BL   @COPIER           Byte in Bitmatrix eintragen
       INC  R13               R13=R13+1
       DEC  R9                Schleifenz{hler um eins erniedrigen
       JNE  NEXT5             Wenn Schleife noch nicht fertig, n{chstes Byte
*
NEXT6  BL   @DSGNCH           Bitmatrix auswerten
       BL   @BRING            Seikosha-String in den Pab-Buffer bringen
       INC  R15               Spaltenz{hler um eins erh|hen
       CI   R15,32            Eine Zeile ausgewertet ?
       JNE  NEXT2             Nein, also n{chste Spalte
*
       BL   @DRUCK            Eine Zeile fertig, also ausdrucken
       DEC  R1Ø               Druckzeilenz{hler um eins erniedrigen
       JNE  NEXT1             Noch nicht alle Zeilen gedruckt
*
       LI   R1,>Ø5ØØ          Alle Zeilen gedruckt, f}nf Bytes zum Drucker
       MOV  @PAB5,RØ
       BLWP @VSBW
*
       MOV  @PABBUF,RØ        Daten f}r Zeilenvorschub laden
       LI   R1,MYDAT2
```

```
       LI   R2,5
       BLWP @VMBW
       BL   @INOUT            Zeilenvorschub ausf}hren
*
       LI   R1,>Ø1ØØ          Eintrittspunkt f}r vorzeitigen Programmabbruch
       MOV  @PAB5,RØ          Ein Byte zum Drucker schicken
       BLWP @VSBW
*
       MOV  @PABBUF,RØ
       LI   R1,>ØFØØ          Textmoduscode in den PAB-Buffer schreiben
       BLWP @VSBW
       BL   @INOUT            Drucker wieder auf Standardmodus umschalten
*
       LI   R1,>ØØØØ          Aus PAB-Buffer kein Byte mehr zum Drucker schicken
       MOV  @PAB5,RØ
       BLWP @VSBW
*
       MOVB @CLOSE,R1         Datei-Schlie~-Befehl in's PAB bringen
       MOV  @PAB,RØ
       BLWP @VSBW
       BL   @INOUT            Datei sclie~en
*
       MOVB @BYTØ,@STATUS     Statusregister l|schen
       LWPI GPLWS             GPL-Arbeitsregister laden
       B    @>ØØ7Ø            N{chste Basic-Anweisung ausf}hren
*
* Unterprogramme
* **************
*
* BRING
*
* Diese Unterprogramm bringt den neuen Seikosha-String
* in den PAB-Buffer.
*
BRING  MOV  R11,@BACK1        Returnadresse sichern
       MOV  @PABBUF,RØ        RØ mit der Adresse des Pabbuffer's laden
       MOV  R15,R12           Wert von R15 nach R12 kopieren
       SLA  R12,3             R12=Wert von R12 mal 8
       A    R12,RØ            R12=R12+Adresse von PABBUF
       LI   R1,STRING         R1 mit Adresse von STRING laden
       LI   R2,8              Acht Byte sind zu }bertragen
       BLWP @VMBW             Byte's in das VDP-RAM schreiben
*
       MOV  @BACK1,R11        Returnadresse wiedergewinnen
       RT                     Zur}ck zum rufenden Programm
*
* DRUCK
*
* Wenn eine komplette Bildschirmzeile f}r den Drucker
* aufbereitet wurde, wird sie durch dieses Unterprogramm
* ausgedruckt und der Drucker f}r die n{chste Zeile vor-
* bereitet.
*
DRUCK  MOV  R11,@BACK1        Returnadresse sichern
       LI   R1,>FFØØ          256 Bytes sollen zum Drucker geschickt werden
       MOV  @PAB5,RØ
       BLWP @VSBW
       BL   @INOUT            Daten zum Drucker schicken
*
```

```
       LI   R1,>Ø5ØØ    F}nf Bytes sollen zum Drucker geschickt werden
       MOV  @PAB5,RØ
       BLWP @VSBW
*
       MOV  @PABBUF,RØ  Daten f}r Druckposition in den PAB-Buffer laden
       LI   R1,MYDAT1
       LI   R2,5        F}nf Bytes sind zu }bertragen
       BLWP @VMBW       Bytes in's VDP-RAM schreiben
       BL   @INOUT      Daten zum Drucker schicken
*
       MOV  @BACK1,R11  Returnadresse wiedergewinnen
       RT               Zur}ck zum rufenden Programm
*
* DSGNCH
*
* Mit Hilfe dieses Unterprogramms wird aus der Bitmatrix der neue
* Seikosha-String berechnet.
*
DSGNCH MOV  R11,@BACK1  Returnadresse sichern
       CLR  RØ          RØ=Ø
       LI   R1,STRING   R1 mit Adresse von STRING laden
       LI   R2,4        4 mal soll die folgende Schleife durchlaufen werden
NEXT7  MOV  RØ,R1       Ein Wort l|schen
       INCT R1          Adresse in R1 um zwei erh|hen
       DEC  R2          Schleifenz{hler um eins erniedrigen
       JNE  NEXT7       Wenn noch nicht alles gel|scht ist n{chstes Wort
*
       LI   R9,8        Schleifenz{hler f}r ein Byte
       LI   R12,STRING  R12=Adresse von STRING
       LI   R15,MATRIX  R15=Adresse der Bitmatrix
       LI   R4,8        Z{hlkonstante f}r aktuelle Bitmatrixadresse
*
NEXT8  MOV  R15,R1Ø     Bitmatrixadresse nach R1Ø kopieren
       LI   R11,6       Z{hler f}r zu berechnende Bits
       LI   RØ,1        RØ mit Verschiebewert laden
       CLR  R8          R8=Ø
       MOVB *R1Ø,R8     Bit Ø von der Bitmatrix nach R8 kopieren
       A    R4,R1Ø      Aktuelle Bitmatrixadresse um acht erh|hen
*
NEXT9  MOVB *R1Ø,R13    N{chstes Bit nach R13 bringen
       SWPB R13         Byte in Wort verwandeln
       SLA  R13,Ø       R13 wird nun um Wert von RØ nach links verschoben
*                       --> R13 mal 2^Wert von RØ
       SWPB R13         Wort in Byte verwandeln
       AB   R13,R8      Die einzelnen Bits zu einem Byte aufaddieren
       A    R4,R1Ø      Aktuelle Bitmatrixadresse um acht erh|hen
       INC  RØ          Verschiebewert um eins erh|hen
       DEC  R11         Bitz{hler um eins erniedrigen
       JNE  NEXT9       Wenn noch keine sieben Bits berechnet, n{chstes Bit
*
       AB   @BYT128,R8  Da das achte Bit bei dem Seikosha GP 1ØØ A immer
*                       auf 1 sein mu~, werden hier 128 zu dem Byte hinzu-
*                       addiert.
       MOVB R8,*R12     Fertiges Byte in den String-Buffer packen
       INC  R15         Urspr}ngliche Bitmatrixadresse um eins erh|hen
       INC  R12         String-Bufferadresse um eins erh|hen
       DEC  R9          Sind acht Bytes berechnet ?
       JNE  NEXT8       Nein, slso n{chstes Byte berechnen
*
       LWPI MYREGI           Ja, also wieder die Arbeitsregister f}r das
*                            Hauptprogramm laden
       MOV  @BACK1,R11       Returnadresse wiedergewinnen
       RT                    Zur}ck zum rufenden Programm
*
* COPIER
*
* Erkl{rungen siehe Unterprogramm 'ZEILE'.
*
COPIER MOV  R11,@BACK2       Returnadresse sichern
*
NEXT1Ø SRL  R5,8             Byte in Wort verwandeln
       SRL  R5,Ø             R5 um Wert von RØ verschieben(dividieren)
       SWPB R5               Wort in Byte verwandeln
       MOVB R5,*R15+         Bit in die Bitmatrix schreiben und Adresse + eins
       SWPB R5               Byte in Wort verwandeln
       SLA  R5,Ø             R5 um Wert von RØ verschieben(multiplizieren)
       SWPB R5               Wort in Byte verwandeln
       SB   R5,R7            Aktuellen Restwert gewinnen
       MOVB R7,R5            Restwert nach R5 kopieren
       DEC  RØ               Verschiebewert um eins erniedrigen
       JNE  NEXT1Ø           Wenn Byte noch nicht zerlegt n{chstes Bit
*
       MOVB R5,*R15+         Bit 7 kann nur noch 1 oder Ø sein
*
       MOV  @BACK2,R11       Returnadresse wiedergewinnen
       RT                    Zur}ck zum rufenden Programm
*
* INOUT
*
* File-Operation durchf}hren
*
INOUT  MOV  R6,@PNTR         L{ngenzeiger setzen
       MOVB @BYTØ,@STATUS    GPL-Statusbyte l|schen
*
       MOVB @>98Ø2,@REGIHB   Zuerst High-Byte und
       MOVB @>98Ø2,@REGILB   dann Low-Byte sichern
       DEC  @REGIHB          High-Byte minus eins
*
       BLWP @DSRLNK          File-Operation durchf}hren
       DATA 8
*
       MOVB @REGIHB,@>9CØ2   Zuerst High-Dyte und
       MOVB @REGILB,@>9CØ2   dann Low-Byte positionieren
*
       RT                    Zur}ck zum rufenden Programmteil
*
* Ende des Programms
*
       END
```

```
100 REM Hardcopyprogramm
           f}r
      SEIKOSHA GP 100 A
110 REM
120 REM POKELISTE f}r das
      Minimemory-Modul
130 REM (c) by
    Bernd Bertling
    Neu-Crengeldanzstra~e 2
    4600 Dortmund 72
140 REM
150 REM Belegter Speicher-
      platz:
      >7118 --> >75AD
      28952 --> 30125
160 DATA 3,32,0,0,0,0,0,18,0
,0,255,0,0,0,96,6
170 DATA 80,73,79,46,67,82,0
,2,0,5,0,9,0,0,10,27
180 DATA 16,0,104,10,10,10,1
0,10,0,128,1,3,0,0,0,0
190 DATA 114,184,0,7,0,0,0,3
2,114,183,1,6,0,32,0,64
200 DATA 114,182,2,5,0,64,0,
96,114,181,3,4,0,96,0,128
210 DATA 114,180,4,3,0,128,0
,160,114,179,5,2,0,160,0,192
220 DATA 114,178,6,1,0,192,0
,192,114,177,7,0,0,224,0,224
230 DATA 114,184,0,7,0,224,1
,0,114,183,1,6,1,0,1,32
240 DATA 114,182,2,5,1,32,1,
64,114,181,3,4,1,64,1,96
250 DATA 114,180,4,3,1,96,1,
128,114,179,5,2,1,128,1,160
260 DATA 114,178,6,1,1,160,1
,160,114,177,7,0,1,192,1,192
270 DATA 114,184,0,7,1,192,1
,224,114,183,1,6,1,224,2,0
280 DATA 114,182,2,5,2,0,2,3
2,114,181,3,4,2,32,2,64
290 DATA 114,180,4,3,2,64,2,
96,114,179,5,2,2,96,2,128
300 DATA 114,178,6,1,2,128,2
,128,114,177,7,0,2,160,2,160
310 DATA 114,184,0,7,2,160,2
,192,114,183,1,6,2,192,2,224
320 DATA 114,182,2,5,2,224,2
,224,114,181,3,0,0,0,0,0
330 DATA 2,224,114,40,2,0,0,
255,200,0,131,12,216,32,113,
64
340 DATA 131,124,2,224,131,2
24,4,32,96,24,0,56,2,224,114
,40
350 DATA 200,32,131,26,113,2
8,192,32,113,24,200,0,131,28
,160,32
360 DATA 113,48,200,0,113,26
,192,32,113,24,2,1,113,30,2,
2
370 DATA 0,16,4,32,96,40,192
,32,113,24,160,32,113,46,2,1
380 DATA 113,28,2,2,0,2,4,32
,96,40,193,160,113,24,161,16
0
390 DATA 113,50,6,160,117,12
8,208,96,113,67,192,32,113,2
4,4,32
400 DATA 96,36,6,160,117,128
,2,1,1,0,192,32,113,26,4,32
410 DATA 96,36,192,32,113,28
,2,1,8,0,4,32,96,36,6,160
420 DATA 117,128,2,1,5,0,192
,32,113,26,4,32,96,36,192,32
430 DATA 113,28,2,1,113,54,2
,2,0,5,4,32,96,40,6,160
440 DATA 117,128,2,10,0,28,2
,8,113,68,2,15,0,0,195,88
450 DATA 5,200,195,152,5,200
,200,24,114,106,5,200,216,24
,114,108
460 DATA 5,136,216,24,114,11
0,5,136,192,13,5,141,4,32,96
,44
470 DATA 9,129,10,49,192,1,1
92,96,113,52,160,1,2,1,114,1
76
480 DATA 2,2,0,8,4,32,96,48,
192,14,5,142,4,32,96,44
490 DATA 9,129,10,49,192,1,1
92,96,113,52,160,1,2,1,114,1
84
500 DATA 2,2,0,8,4,32,96,48,
2,224,114,72,4,193,2,15
510 DATA 114,112,195,160,114
,106,2,13,114,184,210,96,114
,108,210,73
520 DATA 19,11,6,201,210,65,
2,0,0,7,209,94,209,197,6,160
530 DATA 117,94,5,142,6,9,22
,247,210,96,114,110,210,73,1
9,11
540 DATA 6,201,210,65,2,0,0,
7,209,93,209,197,6,160,117,9
4
550 DATA 5,141,6,9,22,247,6,
160,116,254,6,160,116,164,5,
143
560 DATA 2,143,0,32,22,177,6
,160,116,196,6,10,22,158,2,1
570 DATA 5,0,192,32,113,26,4
,32,96,36,192,32,113,28,2,1
580 DATA 113,59,2,2,0,5,4,32
,96,40,6,160,117,128,2,1
590 DATA 1,0,192,32,113,26,4
,32,96,36,192,32,113,28,2,1
600 DATA 15,0,4,32,96,36,6,1
60,117,128,2,1,0,0,192,32
610 DATA 113,26,4,32,96,36,2
08,96,113,66,192,32,113,24,4
,32
620 DATA 96,36,6,160,117,128
,216,32,113,64,131,124,2,224
,131,224
630 DATA 4,96,0,112,200,11,1
14,36,192,32,113,28,195,15,1
0,60
640 DATA 160,12,2,1,114,168,
2,2,0,8,4,32,96,40,194,224
650 DATA 114,36,4,91,200,11,
114,36,2,1,255,0,192,32,113,
26
660 DATA 4,32,96,36,6,160,11
7,128,2,1,5,0,192,32,113,26
670 DATA 4,32,96,36,192,32,1
13,28,2,1,113,54,2,2,0,5
680 DATA 4,32,96,40,6,160,11
7,128,194,224,114,36,4,91,20
0,11
690 DATA 114,36,4,192,2,1,11
4,168,2,2,0,4,192,64,5,193
700 DATA 6,2,22,252,2,9,0,8,
2,12,114,168,2,15,114,112
710 DATA 2,4,0,8,194,143,2,1
1,0,6,2,0,0,1,4,200
720 DATA 210,26,162,132,211,
90,6,205,10,13,6,205,178,13,
162,132
730 DATA 5,128,6,11,22,247,1
78,32,113,65,215,8,5,143,5,1
40
740 DATA 6,9,22,232,2,224,11
4,40,194,224,114,36,4,91,200
,11
750 DATA 114,38,9,133,9,5,6,
197,223,197,6,197,10,5,6,197
760 DATA 113,197,209,71,6,0,
22,245,223,197,194,224,114,3
8,4,91
770 DATA 200,6,131,86,216,32
,113,64,131,124,216,32,152,2
,114,104
780 DATA 216,32,152,2,114,10
5,6,32,114,104,4,32,96,56,0,
8
790 DATA 216,32,114,104,156,
2,216,32,114,105,156,2,4,91,
0,0
800 CALL INIT
810 REM
820 REM Schreibschleife
830 REM
840 REM Die Variable 'T'
soll Ihnen eine Sicherheit
geben, die Data's richtig
eingegeben zu haben.
850 REM Als Pr}fsumme mu~
      73194 herauskommen.

860 FOR X=28952 TO 29223 STE
P 16
870 READ A,B,C,D,E,F,G,H,I,K
,L,M,N,O,P,Q
880 T=T+A+B+C+D+E+F+G+H+I+K+
L+M+N+O+P+Q
890 CALL LOAD(X,A,B,C,D,E,F,
G,H,I,K,L,M,N,O,P,Q)
900 NEXT X
910 FOR X=29376 TO 30127 STE
P 16
920 READ A,B,C,D,E,F,G,H,I,K
,L,M,N,O,P,Q
930 T=T+A+B+C+D+E+F+G+H+I+K+
L+M+N+O+P+Q
940 CALL LOAD(X,A,B,C,D,E,F,
G,H,I,K,L,M,N,O,P,Q)
950 NEXT X
960 REM Def-Pointer setzen
970 CALL LOAD(28700,113,24,1
27,248)
980 REM Def-Table setzen
990 CALL LOAD(32760,72,67,79
,80,89,32,114,192)
1000 END
```

CAR/CAD

Das Programm gestattet die Erstellung von kurzen Trickfilmabläufen. Die erstellten Character Strings werden entweder als MERGE Datei oder als Editor/Assembler File abgespeichert.
Mit der MERGE Option des Extended Basic können die erstellten Strings dann in beliebige Basic-Programme übernommen werden. Mit dem Assembler/Editor können die Strings auch in Quelldateien eingebunden werden.
Die Merge Datei kann mit Merge DSK1.Char Mem übernommen werden und das E/A File mit Dsk1.Charcode.
Zur Bedienung:
Nachdem das Basicprogramm geladen und gestartet ist, beginnt der Bildschirmaufbau. Zuerst läuft eine kleine Demonstration ab. Danach wird das Zeichenfeld gelöscht und es erscheint der Sprite/Cursor.
Wird jetzt der Joystick nicht benutzt, und auch keine Taste gedrückt, springt das Zeichenprogramm "Draw" selbsttätig in das Basicprogramm zurück.
Die Geschwindigkeit, mit der sich der Cursor bewegt und die Zeit, die vergeht bis das Programm zurückspringt, wird im Call Link Aufruf festgelegt.
Das "Select" Programm ermöglicht jetzt die Auswahl eines Befehles.
Das "Select" Programm springt erst bei Betätigung der Feuertaste wieder zum Basic Programm zurück.
Sowohl bei "Draw" als auch bei "Select" werden bei einem Rücksprung zum Basic-Programm die aktuelle Zeile und Spalte der Cursorposition an das aufrufende Programm übergeben.
Die Programmsegmente "Draw", "Select" und "Scroll" sind außerdem ohne weitere Probleme auch in andere Programme einzubinden.
Ich hoffe, daß dies als Anleitung zum Programm genügt, wenn man ein wenig mit dem Programm spielt, versteht man es wohl am schnellsten.

Eberhard Schael

Erklärungen zu den Programmsegmenten:

@CCAD/DEF#
Enthält alle für das Maschinenprogramm wichtigen Variablenspeicher, REF/DEF Table, Zeichendefinitionen sowie die Copy Direktiven für die restlichen Files.

@CHARSCAN#
Dieses Programmsegment tastet das Zeichenfeld ab und liefert die Variable VAR$ an das TI-Basic-Programm zurück.
Der Aufruf aus dem Basic erfolgt mit Call Link (CHRSCN",VAR$).

@DSPACPT#
Entspricht dem Programm aus 99 Special II, jedoch wurde z.B. auf modulunabhängige Routinen verzichtet.
Aufruf über
Call Link ("DSPLAY", Zeile, Spalte, Stringlänge, VAR§)
Call Link ("ACCEPT", Zeile, Spalte, Stringlänge, VAR$)

@INVERT#
Tauscht im Zeichenfeld das Leerzeichen (ASCII 32) gegen dieses Zeichen (ASCII 126), das ist die Tilde (~), aus, und umgekehrt.
Aufruf über Call Link ("INVERT")

@SCROLL#
Diese Routine scrollt ein im Call Link Statement definiertes Feld in eine von 4 vorgegebenen Richtungen.
Aufruf (auch aus anderen TI-Basic-Programmen) über Call Link ("SCROLL", ZEMIN; SPMIN, ZEMAX, SPMAX, Richtung, Scrollcount)
Das durch ZEMIN, SPMIN, ZEMAX, SPMAX definierte Feld wird nach unten =1, nach oben =2, nach links =3 und nach rechts =4 gescrollt. Scrollcount gibt an, wie oft dieser Vorgang wiederholt wird.

@SCRMEMO#
Call Link ("SCRSAV")
Diese Routine speichert beim Aufruf den kompletten Bildschirminhalt.
Call Link ("SCRRET")
Bringt den zuvor gespeicherten Bildschirm wieder zurück.

@SELECT#
Diese Routine bewegt in dem – im Call Link Statement – definierten Zeichenfeld den Cursor. Sobald die Feuertaste gedrückt wird, springt die Routine zurück zum Basic-Programm.
Die augenblickliche Position des Cursors (Zeile + Spalte) wird dabei den Variablen Value 1 und Value 2 zugewiesen.
Aufruf über
Call Link ("SELECT", ZEMIN, SPMIN,

ZEMAX, SPMAX, ZEPOS, SSPOS, Delaytime, Value 1, Value 2).

@DRAW#
Aufruf der Routine über Call Link ("DRAW", ZEMIN, SPMIN, ZEMAX, SPMAX, ZEPOS, SPPOS, Delaytime, Returntime, Value 1, Value 2).
Wenn die Routine zurückspringt in das Basic-Programm wird die Position des Cursors (Zeile + Spalte) an die Variablen Value 1 + Value 2 übermittelt.

@DRAW#
Dieses ist die Zeichenroutine.
Aufruf über
Call Link ("DRAW", ZEMIN, SPMIN, ZEMAX, SPMAX, ZEPOS, SPPOS, Delaytime, Returntime)
Die ersten vier Parameter definieren auch hier wieder das Zeichenfeld. ZEPOS & SPPOS geben die Position an, wo bei Aufruf der Routine ein Sprite erscheint.
Dieses Sprite ist ein Cursor, der die augenblickliche Schreibposition markiert.

Bewegt wird Cursor mit dem Joystick 1. Wird die Feuertaste gedrückt, dann erscheint an dieser Stelle eine Tilde (ASCII 126). Ohne die Feuertaste zu drücken, löscht das Kreuz die betreffende Schreibposition.
Delaytime bestimmt die Geschwindigkeit des Cursors und kann zwischen 1 – 2000 liegen.
Werden Joystick und Feuertaste nicht benutzt, springt die Zeichenroutine bei Erreichen von Returntime zurück in das Basic-Programm.

@SUBPROG#
Dieses Segment enthält alle mehrfach benutzten Routinen.

Zum Basic-Programm CHAR/CAD:

Das Programm ermöglicht das Erstellen von Charactercodes für max. 4 Schreibstellen gleichzeitig. Springt der Cursor aus dem Zeichenfeld zur Auswahlliste, wählt man mit dem Joystick den entsprechenden Befehl. Sobald die Zeichenroutine wieder ins Basic springt (keine Taste drücken, Joystick 1 nicht bewegen!), wird der entsprechende Befehl ausgeführt.
Da es lästig ist, gespeicherte Characterstrings abzutippen oder als Datei in Programme einzubinden, speichert das Programm die erstellten Strings als Merge-Datei unter dem Namen "CHARMEM" auf der Diskette.
Unter Zuhilfenahme des EX-Basic-Moduls kann die Datei – durch Merge "DSK 1.CHARMEM" – in ein anderes Programm übernommen werden.
Außerdem ermöglicht der Befehl "TRICKFILM" kleine Bildfolgen mit Trickfilmcharacter, sofern mehrere Codes erstellt werden.

Anmerkung: Zu dem Programm müssen die Utilities BSCSUB des E/A Moduls geladen werden.

Benötigte Ausstattung:

Konsole, E/A Modul, P-Box mit 32 K RAM, 1 Diskettenstation, Joystick 1 und EX-BASIC Modul

```
**********************************
*     EBERHARD SCHAEL            *
*     HAGENER STR. 15            *
*     5BØØ HAGEN 1               *
*     Ø2331/6B7154               *
**********************************
*BPROGRAMMBEGMENT "@CCAD/DEF#"
*LAENGE=129 ZEILEN
*
       DEF  DRAW,CHRSCN,INVERT
       DEF  DSPLAY,ACCEPT,SCROLL
       DEF  SCRBAV,BCRRET,SELECT
*
       REF  VSBW,VMBW,VMBR,VSBR
       REF  KSCAN,BTRASG,STRREF
       REF  NUMASG,NUMREF,ERR,XMLLNK
*
MYWS   BSS  >2Ø       EIGENE ARBEITSREGISTER
RØHB   EQU  MYWS      ABSOLUTE ADREBSEN FUER
RØLB   EQU  MYWS+1    ARBEITSREGISTER RØ
*
       EVEN
STRING BSS  >41       BUFFER FUER CHARACTERSTRING
DABTRG BSB  >21       BUFFER FUER DISPLAY&ACCEPT STRING
SCREEN BSS  >3ØØ      BUFFER FUER EINEN SCHIRMINHALT =76B CHARACTERS
*
CHRBUF BSS  4         BUFFER UND FLAGS FUER
FLAGØ1 BBB  2         DAB PROGRAMMSEGMENT
FLAGØ2 BSS  2         CHARBCAN
CHRBYT BSS  2
WRDBUF BSS  2
BCRCHR BSS  1
SCRBUF BSS  32
*
ROW    BSS  >2        BUFFER FUER ZEILE
COL    BSS  >2        BUFFER FUER SPALTE
PRMTR  BSS  >2        LINK PARAMETER BUFFER
ZEMAX  BSS  >2        BUFFER FUER DIE HOECHSTE ZEILE
ZEMIN  BBB  >2          "     "    "  NIEDRIGSTE ZEILE
SPMAX  BSB  >2          "     "    "  HOECHSTE SPALTE
SPMIN  BSS  >2          "     "    "  NIEDRIGSTE SPALTE
*
RTLEV1 BSS  >2        DIVERSE BUFFER FUER
RTLEV2 BSS  >2        RETURNADRESSEN IN VER-
RTLEV3 BSS  >2        SCHIEDENEN LEVELN
TACT   BSS  >2        BUFFER FUER T1me•ACTue11
NEWPOS BSS  >2        BUFFER FUER NEUE POSITION
ZEPOS  BSB  >2        BUFFER FUER ZEILE UND
SPPOS  BSS  >2        BPALTE
DLYTME BSS  >2        BUFFER FUER DELAYTIME
TVRG   BSS  >2        BUFFER FUER VORGABEZEIT
BCRDIR BSB  >2        BUFFER FUER SCROLL-RICHTUNG
SPDIF  BSS  >2        BUFFER FUER DIFFERENZBETRAG AUS SPMAX UND BPMIN
ZEDIF  BSS  >2          "      "       "              "  ZEMAX  "  ZEMIN
BCRCNT BSB  >2        BUFFER FUER SCROLLCOUNT
*
HØØ    DATA ØØØØ      WORTKONSTANTE
H29    DATA 29
H3Ø    DATA 3Ø
```

```
CHARØ0 DATA >8000
VALUE1 DATA >0001
VALUE2 DATA >0002
*
*HEXDTA ENTHAELT DIE MOEGLICHEN KOMBINATIONEN
*FUER DIE ABTASTUNG DES ZEICHENFELDES.
*
HEXDTA DATA >8080,>8080,>8080,>80DE
       DATA >8080,>DE80,>8080,>DEDE
       DATA >80DE,>8080,>80DE,>80DE
       DATA >80DE,>DE80,>80DE,>DEDE
       DATA >DE80,>8080,>DE80,>80DE
       DATA >DE80,>DE80,>DE80,>DEDE
       DATA >DEDE,>8080,>DEDE,>80DE
       DATA >DEDE,>DE80,>DEDE,>DEDE
*
*HXBYTE ENTHAELT DIE ASCII-WERTE FUER 0-9&A-F
*
HXBYTE BYTE >30,>31,>32,>33,>34,>35
       BYTE >36,>37,>38,>39,>41,>42
       BYTE >43,>44,>45,>46
*
BYT1   BYTE 1      BYTE KONSTANTE
BYT28  BYTE 28
BYT96  BYTE 96
BYTM96 BYTE -96
*
       EVEN
JYUP   BYTE  4,0   *JOYST HOCH
JYRT   BYTE  0,4   *JOYST RECHTS
JYDN   BYTE  -4,0  *JOYST RUNTER
JYLT   BYTE  0,-4  *JOYST LINKS
*
JYLTDN BYTE  -4,-4 *JOYST LINKS&RUNTER
JYLTUP BYTE  4,-4  *JOYST LINKS&HOCH
JYRTDN BYTE  -4,4  *JOYST RECHTS&RUNTER
JYRTUP BYTE  4,4   *JOYST RECHTS&HOCH
*
HEXFF  BYTE  >FF     KEINE TASTE
KEY18  BYTE  18      TASTE Q ODER FEUERKNOPF
CLEAR  BYTE  2         "   CLEAR
ERASE  BYTE  7         "   ERASE
LEFT   BYTE  8         "   LEFT
RIGHT  BYTE  9         "   RIGHT
ENTER  BYTE  13        "   ENTER
CRSR   BYTE  126       CURSOR CHARACTER
*
       EVEN
FAC    EQU   >834A   FPOINT AKKUMULATOR
STATUS EQU   >837C   GPL STATUS BYTE
KEYBRD EQU   >8375   ASCII WERT DER GEDRUECKTEN TASTE
JOYY   EQU   >8376   JOYSTICK WERTE
JOYX   EQU   >8377
GPLWS  EQU   >83E0   GPL ARBEITSREGISTER
*
ERRBA  EQU   >1600   BAD ARGUMENT
CFI    EQU   >1200   REAL-INTEGER UEBERSETZUNG
CIF    EQU   >2300   INTEGER-FPOINT UEBERSETZUNG
*
```

```
CLRDTA DATA >8080,>8080,>8080,>8080   32 LEERZEICHEN
       DATA >8080,>8080,>8080,>8080
       DATA >8080,>8080,>8080,>8080
       DATA >8080,>8080,>8080,>8080
SPTCHR DATA >0000,>1010,>7C10,>1000   SPRITE CHARACTER  ("+")
SPTDTA DATA >2020,>8B0F,>D000         WERTE FUER SPRITE AUFRUF
*
       COPY "DSK1.@SUBPROG#"          COPY DIREKTIVEN FUER ALLE
       COPY "DSK1.@DRAW#"             ANDEREN PROGRAMMTEILE
       COPY "DSK1.@CHARSCAN#"
       COPY "DSK1.@INVERT#"
       COPY "DSK1.@DSPACPT#"
       COPY "DSK1.@SCROLL#"
       COPY "DSK1.@SCRMEMO#"
       COPY "DSK1.@SELECT#"
*
       END
```

```
*PROGRAMMSEOMENT "@INVERT#"
*INVERTIERT DAS ZEICHENFELD
*LAENGE=42 ZEILEN
*
INVERT LWPI MYWS              EIGENE ARBEITSREGISTER LADEN
*
       CLR  R3                R3=HILFSREOISTER FUER SPALTENPOSITION
       CLR  R4                R4=HILFSREGISTER FUER ZEILENPOSITION
       LI   RØ,>ØØ22          ANFANGSADRESSE IM VDP-RAM
       CLR  R1                BYTE LESEN
INVLP1 BLWP @VSBR
       NOP
       C    R1,@CHAR8Ø        MSB R1=8Ø
       JEQ  CHRSTØ            DANN  ASCII 32 > < GEGEN
*                             ASCII 126 >~< TAUSCHEN
       BL   @CHRST1           MSB R1=DE
*                             ASCII 126>~< GEGEN
*                             ASCII 32 > < TAUSCHEN
       JMP  NXTLP1
CHRSTØ BL   @CHRST2
NXTLP1 INC  RØ                NAECHSTE SPALTE
       INC  R3                HILFSREGISTER FUER SPALTENPOSITION
       CI   R3,>ØØ1Ø          16 SPALTEN GEPRUEFT?
       JNE  INVLP1
       AI   RØ,>ØØ1Ø          NAECHSTE ZEILE
       CLR  R3                HILFSREGISTER LOESCHEN
       INC  R4                HILFSREGISTER ZEILE ERHOEHEN
       CI   R4,>ØØ1Ø          ALLE ZEILEN BEARBEITET?
       JNE  INVLP1            WENN NICHT-WEITER
       BL   @BSRTN            ZURUECK ZU BASIC
*
CHRST1
       LI   R1,>8ØØØ          LEERZEICHEN SETZEN
       BLWP @VSBW
       NOP
       RT
*
CHRST2
       LI   R1,>DEØØ          TILDE >~< SETZEN
       BLWP @VSBW
       NOP
       RT
```




```
*PROORAMMSEOMENT "@SCROLL#"
*LAENOE=3ØØ ZEILEN
*
SCROLL LWPI MYWS              EIGENE REGISTER LADEN
*
       LI   R1,5              SCROLLRICHTUNG IST 5.LINK PARAMETER
       MOV  R1,@PRMTR
       BL   @OETVAL           WERT OEWINNEN
       DATA 1,4,SCRDIR        UND IN SCRDIR SPEICHERN
*
       CLR  @FLAOØ1           FLAGØ1 LOESCHEN
       INC  @FLAOØ1           FLAOØ1 =1
       C    @SCRDIR,@FLAGØ1   RICHTUNO=1?
       JNE  SCR2              NEIN--> SCR2
       BL   @SCRVL1           SCROLLPARM. GEWINNEN
       BL   @SCDN             UPRO. SCROLLDOWN
       BL   @BSRTN            BASIC RETURN
*
SCR2   INC  @FLAGØ1           FLAGØ1=2
       C    @SCRDIR,@FLAOØ1   RICHTUNG=2?
       JNE  SCR3              NEIN--> SCR3
       BL   @SCRVL1           SCROLLPARM.GEWINNEN
       BL   @SCUP             UPRO.SCROLLUP
       BL   @BSRTN            BASIC RETURN
*
SCR3   INC  @FLAOØ1           SCROLLRICHTUNG 3 =RECHTS
       C    @SCRDIR,@FLAOØ1
       JNE  SCR4
       BL   @SCRVL2
       BL   @SCRT
       BL   @BSRTN
*
SCR4   BL   @SCRVL2           SCROLLRICHTUNO 4 =LINKS
       BL   @SCLT
       BL   @BSRTN
*
SCDN   MOV  R11,@RTLEV3       RETURNADRESSE GEWINNEN
       LIMI Ø
       BL   @SCVLC1           SCROLL WERTE KONTROLLE
*
       CLR  @FLAGØ2
       MOV  @SPMIN,@SPPOS
*
DNLP1  CLR  @FLAOØ1
       MOV  @ZEMAX,@ZEPOS
       DEC  @ZEPOS
DNLP2  BL   @VALCLC           ERSTE POSITION IN VDP-RAM
       MOV  @NEWPOS,RØ        BERECHNEN
       LI   R1,SCRBUF
       MOV  @SPDIF,R2         SPMAX-SPMIN ERGIBT SPDIF
       BLWP @VMBR             EINE ZEILE LESEN
       INC  @ZEPOS            UND DIE OELESENE ZEILE
       BL   @VALCLC           EINE ZEILE TIEFER
       MOV  @NEWPOS,RØ        WIEDER IN DAS VDP-RAM
       LI   R1,SCRBUF         SCHREIBEN.
       MOV  @SPDIF,R2
       BLWP @VMBW
       DECT @ZEPOS            NAECHSTE ZEILE LESEN
       INC  @FLAOØ1           ZEMAX-ZEMIN=ZEDIF
```

```
       C    @FLAGØ1,@ZEDIF  ALLE ZEILEN BEARBEITET?
       JNE  DNLP2           WENN NICHT DANN WEITER
*
       MOV  @ZEMIN,@ZEPOS   LETZTE ZEILE
       BL   @VALCLC         BERECHNEN
       MDV  @NEWPDS,RØ      UND DANN MIT LEER-
       LI   R1,CLRDTA       ZEICHEN AUS CLRDTA
       MDV  @SPDIF,R2       UEBERSCHREIBEN
       BLWP @VMBW
       INC  @FLAGØ2
       C    @FLAGØ2,@SCRCNT SCRDLLVDRGANG ENTSPRECHEND
       JNE  DNLP1           OFT WIE VORGEGEBEN WIEDERHOLT?
*
       MDV  @RTLEV3,R11     RETURNADRESSE GEWINNEN UND
       LIMI 2
       RT                   ZURUECK
*****************************************
*SCUP BEINHALTET DAS GLEICHE PRDGRAMM     *
*WIE UNTER SCDN BESCHRIEBEN.              *
*DER EINZIGE UNTERSCHIED LIEGT DARIN      *
*DAS DIE ZEILEN NACH OBEN GESCHDBEN WERDEN*
*****************************************
SCUP   MOV  R11,@RTLEV3
       LIMI Ø
       BL   @SCVLC1
*
       CLR  @FLAGØ2
       MDV  @SPMIN,@SPPDS
*
UPLP1  CLR  @FLAGØ1
       MDV  @ZEMIN,@ZEPDS
       INC  @ZEPDS
UPLP2  BL   @VALCLC
       MOV  @NEWPDS,RØ
       LI   R1,SCRBUF
       MDV  @SPDIF,R2
       BLWP @VMBR
       DEC  @ZEPDS
       BL   @VALCLC
       MOV  @NEWPOS,RØ
       LI   R1,SCRBUF
       MDV  @SPDIF,R2
       BLWP @VMBW
       INCT @ZEPDS
       INC  @FLAGØ1
       C    @FLAGØ1,@ZEDIF
       JNE  UPLP2
*
       MDV  @ZEMAX,@ZEPOS
       BL   @VALCLC
       MOV  @NEWPOS,RØ
       LI   R1,CLRDTA
       MOV  @SPDIF,R2
       BLWP @VMBW
       INC  @FLAGØ2
       C    @FLAGØ2,@SCRCNT
       JNE  UPLP1
*
       MOV  @RTLEV3,R11
```

```
       LIMI 2
       RT
*
SCRT   MOV  R11,@RTLEV3     RETURNADRESSE GEWINNEN
       LIMI Ø
       BL   @SCVLC2

       CLR  @FLAGØ2
*
RTLP1  CLR  @FLAGØ1
       MOV  @ZEMIN,@ZEPDS   ANFANGSPOSITIDN
       MOV  @SPMIN,@SPPDS   IM VDP-RAM
RTLP2  BL   @VALCLC         BERECHNEN
       MDV  @NEWPDS,RØ      EINE ZEILE
       LI   R1,SCRBUF       NACH SCRBUF UEBERTRAGEN
       MOV  @SPDIF,R2       LAENGE DER ZEILE WIEDER
       BLWP @VMBR           SPMAX-SPMIN=SPDIF
       LI   R1,>8ØØØ        LEERZEICHEN AN DIE
       BLWP @VSBW           ANFANGSADRESSE SCHREIBEN
       INC  RØ              SPALTE=SPALTE+1
       LI   R1,SCRBUF       SCRBUF WIEDER IN DAS
       MOV  @SPDIF,R2       VDP-RAM SCHREIBEN
       BLWP @VMBW
       INC  @ZEPOS          NAECHSTE ZEILE
       INC  @FLAGØ1
       C    @FLAGØ1,@ZEDIF  ALLE ZEILEN VERARBEITET?
       JNE  RTLP2           NEIN-->WEITER
       INC  @FLAGØ2
       C    @FLAGØ2,@SCRCNT SCROLLCOUNT ERREICHT?
       JNE  RTLP1           NEIN-->WEITER
*
       MOV  @RTLEV3,R11     RETURNADRESSE GEWINNEN
       LIMI 2
       RT                   UND ZURUECK
*****************************************
*SCLT BEINHALTET DAS GLEICHE PROGRAMM     *
*WIE IN SCRT BESCHRIEBEN.AUCH HIER LIEGT DER*
*UNTERSCHIED NUR IN DER RICHTUNG IN       *
*DER DIE ZEILEN GESCHDBEN WERDEN.         *
*****************************************
SCLT   MOV  R11,@RTLEV3
       LIMI Ø
       BL   @SCVLC2
*
       CLR  @FLAGØ2
*
LTLP1  CLR  @FLAGØ1
       MOV  @ZEMIN,@ZEPOS
       MOV  @SPMIN,@SPPOS
       INC  @SPPOS
LTLP2  BL   @VALCLC
       MOV  @NEWPOS,RØ
       LI   R1,SCRBUF
       MOV  @SPDIF,R2
       BLWP @VMBR
       DEC  @SPPDS
       BL   @VALCLC
       MOV  @NEWPOS,RØ
       LI   R1,SCRBUF
```

```
       MOV  @SPDIF,R2
       BLWP @VMBW
       MOV  @SPMAX,@SPPOS
       BL   @VALCLC
       MOV  @NEWPOS,R0
       LI   R1,>8000
       BLWP @VSBW
       MOV  @SPMIN,@SPPOS
       INC  @SPPOS
       INC  @ZEPOS
       INC  @FLAG01
       C    @FLAG01,@ZEDIF
       JNE  LTLP2
       INC  @FLAG02
       C    @FLAG02,@SCRCNT
       JNE  LTLP1
*
       MOV  @RTLEV3,R11
       LIMI 2
       RT
*
SCVLC1 C    @ZEMAX,@ZEMIN   ZEMAX >ZEMIN?
       JGT  CON1            JA
       B    @BADARG         NEIN -->"BAD ARGUMENT " UND STOP
CON1   C    @SPMAX,@SPMIN   SPMAX GROESSER ODER GLEICH SPMIN
       JGT  CON2
       JEQ  CON2
       B    @BADARG         WENN NICHT DANN "BAD ARGUMENT" UND STOP
*
CON2   MOV  @SPMAX,@SPDIF   SPALTENDIFFERENZ
       S    @SPMIN,@SPDIF   BILDEN
       INC  @SPDIF
*
       MOV  @ZEMAX,@ZEDIF   ZEILENDIFFERENZ
       S    @ZEMIN,@ZEDIF   BILDEN
       INCT @ZEDIF
       C    @ZEDIF,@SCRCNT  SCROLLCOUNT AN
       JGT  VLCRT1          ZEILENDIFFERENZ
       MOV  @ZEDIF,@SCRCNT  ANPASSEN
       DEC  @SCRCNT
*
VLCRT1 DECT @ZEDIF
       RT
*
SCVLC2 C    @ZEMAX,@ZEMIN   ZEMAX GROESSER ODER GLEICH ZEMIN
       JGT  CON11
       JEQ  CON11
       B    @BADARG         "BAD ARGUMENT" UND STOP
CON11  C    @SPMAX,@SPMIN   SPMAX GROESSER SPMIN
       JGT  CON22
       B    @BADARG         "BAD ARGUMENT"UND STOP
*
CON22  MOV  @ZEMAX,@ZEDIF   ZEILENDIFFERENZ BILDEN
       S    @ZEMIN,@ZEDIF
       INC  @ZEDIF
*
       MOV  @SPMAX,@SPDIF   SPALTENDIFFERENZ BILDEN
       S    @SPMIN,@SPDIF
       INCT @SPDIF
```

```
       C    @SPDIF,@SCRCNT  SCROLLCOUNT  AN
       JGT  VLCRT2          SPDIFFERENZ ANPASSEN
       MOV  @SPDIF,@SCRCNT
       DEC  @SCRCNT
*
VLCRT2 DECT @SPDIF
       RT
***************************
*PARAMETER-UEBERNAHME FUER*
*SCROLLDOWN UND SCROLLUP  *
***************************
SCRVL1 MOV  R11,@RTLEV2
       LI   R1,1
       MOV  R1,@PRMTR
       BL   @GETVAL
       DATA 1,23,ZEMIN
*
       INC  @PRMTR
       BL   @GETVAL
       DATA 1,32,SPMIN
*
       INC  @PRMTR
       BL   @GETVAL
       DATA 2,24,ZEMAX
*
       INC  @PRMTR
       BL   @GETVAL
       DATA 1,32,SPMAX
*
       INCT @PRMTR
       BL   @GETVAL
       DATA 1,24,SCRCNT
*
       MOV  @RTLEV2,R11
       RT
****************************
*PARAMETER-UEBERNAHME FUER *
*SCROLLRIGHT UND SCROLLLEFT*
****************************
SCRVL2 MOV  R11,@RTLEV2
       LI   R1,1
       MOV  R1,@PRMTR
       BL   @GETVAL
       DATA 1,24,ZEMIN
*
       INC  @PRMTR
       BL   @GETVAL
       DATA 1,32,SPMIN
*
       INC  @PRMTR
       BL   @GETVAL
       DATA 1,24,ZEMAX
*
       INC  @PRMTR
       BL   @GETVAL
       DATA 1,32,SPMAX
*
       INCT @PRMTR
       BL   @GETVAL
```

```
*DAS PROGRAMMSEGMENT "@SELECT#"
*ENTHAELT DIE ZEICHENROUTINE
*LAENGE=73 ZEILEN
*
SELECT LWPI MYWS
*
       BL    @SLCTVL          PARAMETER HOLEN
       BL    @SPTSET          "CALL SPRITE...."
       BL    @VALTST          ZEICHENPARAMETER PRUEFEN
       BL    @VALCLC          POSITION BERECHNEN
       BL    @SPTLCT          "CALL LOCATE..."
*
SELCT1 BL     @DELAY          PAUSE
       LI     RØ,>Ø1ØØ        KEYBOARD LINKE SEITE&JOYST.1
       MOVB   RØ,@>8374
       BLWP   @KSCAN          TASTATUR ABFRAGEN
       C      @JOYY,@HØØ      JOYSTICK NICHT BEWEGT
       JNE    SELCT3
*
       CB     @KEYBRD,@KEY18  FEUERTASTE GEDRUECKT
       JEQ    SELCT2          JA,DANN ZURUECK
       JNE    SELCT1          NEIN,WIEDER VON VORNE
*
SELCT2 BL     @RTVAL1         ZEILE UND SPALTE AN LINK
       BL     @BSRTN          UEBERGEBEN UND ZURUECK ZUM BASIC
*
SELCT3 BL     @JOYCMP         JOYSTICK AUSWERTEN
       BL     @VALTST         WERTE TESTEN
       BL     @VALCLC         POSITION BERECHNEN
       BL     @SPTLCT         "CALL LOCATE..."
       BL     @DELAY          PAUSE
       JMP    SELCT1
****************************
*SLCTVAL UEBERNIMMT DIE    *
*PARAMETER FUER DAS SELECT *
*PROGRAMM.                 *
****************************
SLCTVL MOV   R11,@RTLEV2
       LI    R1,1
       MOV   R1,@PRMTR
       BL    @GETVAL
       DATA 1,24,ZEMIN
       DEC   @ZEMIN
*
       INC   @PRMTR
       BL    @GETVAL
       DATA 1,32,SPMIN
       DEC   @SPMIN
*
       INC   @PRMTR
       BL    @GETVAL
       DATA 1,24,ZEMAX
       INC   @ZEMAX
*
       INC   @PRMTR
       BL    @GETVAL
       DATA 1,32,SPMAX
       INC   @SPMAX
*
       INC   @PRMTR
       BL    @GETVAL
       DATA 1,24,ZEPOS
*
       INC   @PRMTR
       BL    @GETVAL
       DATA 1,32,SPPOS
*
       INC   @PRMTR
       BL    @GETVAL
       DATA 1Ø,2ØØØ,DLYTME
*
       MOV   @RTLEV2,R11
       RT
```

```
*PROGRAMMSEGMENT "@SUBPROG#"
*ENTHAELT ALLE MEHRFACH
*GENUTZTEN UNTERPROGRAMME
*LAENGE=130 ZEILEN
*
DELAY  MOV  @DLYTME,15    HOLT WERT FUER DELAY NACH R15
DELYLP NOP                TUT NICHTS
       DEC  15            R15=R15-1
       MOV  15,15         R15=0?
       JNE  DELYLP        WENN NICHT,DANN WEITERZAEHLEN
       RT
******************************************
*VALCLC ERRECHNET AUF DER BASIS          *
*VON ZEILE(ZEPOS)&SPALTE(SPPOS) NACH DER *
*FORMEL (ZEILE-1*32)+SPALTE-1 DIE        *
*NEUE POSITION IM VDP-RAM                *
******************************************
VALCLC MOV  @ZEPOS,R0    ZEPOS NACH R0
       DEC  R0           ZEPOS=ZEPOS-1
       SLA  R0,5         *32
       A    @SPPOS,R0    +SPPOS
       DEC  R0           -1
       MOV  R0,@NEWPOS   NEUE POSITION NACH NEWPOS(ition
       RT
*****************************************************
*GETVAL  UBERNIMMT PARAMETER UND UEBERPRUEFT        *
*OB SIE MIT DEN ANGEGEBENEN GRENZEN UEBEREINSTIMMEN *
*****************************************************
*
GETVAL MOV  *R11+,R2      UNTERE GRENZE
       MOV  *R11+,R3      OBERE  GRENZE
       MOV  *R11+,R4      ZIELADRESSE
       MOV  R11,@RTLEV1   RETURNADRESSE SICHERN
*
       DEC  R2            GRENZEN FUER VERGLEICH
       INC  R3            AUFBEREITEN
*
       CLR  R0            LINK  PARAMETER NUMMER
       MOV  @PRMTR,R1     HOLEN
       BL   @GETPAR       WERT HOLEN UND IN
       MOV  @FAC,*R4      ZIEL SPEICHERN
*
       C    *R4,R2        UNTERE GRENZE TESTEN
       JGT  TSTOG         OK.
       B    @BADARG       GRENZE VERLETZT
TSTOG  C    *R4,R3        OBERE GRENZE TESTEN
       JLT  GPRT          OK.
       B    @BADARG       GRENZE VERLETZT

GPRT   MOV  @RTLEV1,R11   RETURNADRESSE WIEDERGEWINNEN
       RT                 UND AB DIE POST ZURUECK
*
GETPAR BLWP @NUMREF       NUMERISCHEN PARAMETER
       BLWP @XMLLNK       UEBERNEHMEN UND IN INTEGER
       DATA CFI           TRANSFORMIEREN
       RT                 ZURUECK
*
BADARG LI   R0,ERRBA      FEHLERCODE NACH HBYTE R0
       BLWP @ERR          ANZEIGE "BAD ARGUMENT" UND STOP
```

```
*
BSRTN  MOV  @H00,@STATUS  STATUS LOESCHEN
       LWPI GPLWS         GPL ARBEITSREGISTER LADEN
       B    @>0070        ZURUECK ZUM BASIC
**********************************
*JOYCoMPare VERGLEICHT DIE WERTE *
*FUER JOYY UND JOYX UND ERMITTELT*
*DIE JEWEILIGE RICHTUNG.         *
**********************************
JOYCMP C     @JOYY,@JYUP    JOYSTICK NACH OBEN?
       JEQ   MVUP           JA!
       C     @JOYY,@JYRT    JOYSTICK NACH RECHTS?
       JEQ   MVRT           JA!
       C     @JOYY,@JYDN    UND IMMER SO WEITER...
       JEQ   MVDN
       C     @JOYY,@JYLT
       JEQ   MVLT
       C     @JOYY,@JYLTUP
       JEQ   MVLTUP
       C     @JOYY,@JYLTDN
       JEQ   MVLTDN
       C     @JOYY,@JYRTUP
       JEQ   MVRTUP
       C     @JOYY,@JYRTDN
       JEQ   MVRTDN
       JMP   JOYRTN
***********************
MVUP   DEC   @ZEPOS        NACH OBEN--> ZEILE=ZEILE-1
       JMP   JOYRTN        ZURUECK
*
MVDN   INC   @ZEPOS        NACH UNTEN-->ZEILE=ZEILE+1
       JMP   JOYRTN        ZURUECK
*
MVRT   INC   @SPPOS        NACH RECHTS-->SPALTE=SPALTE+1
       JMP   JOYRTN        ZURUECK
*
MVLT   DEC   @SPPOS        NACH LINKS-->SPALTE=SPALTE-1
       JMP   JOYRTN
*
MVRTUP INC   @SPPOS        NACH RECHTS OBEN-->SPALTE=SPALTE+1
       DEC   @ZEPOS        ZEILE=ZEILE-1
       JMP   JOYRTN        ZURUECK
*
MVRTDN INC   @SPPOS        NACH RECHTS UNTEN-->SPALTE=SPALTE+1
       INC   @ZEPOS        ZEILE=ZEILE+1
       JMP   JOYRTN        ZURUECK
*
MVLTUP DEC   @ZEPOS        NACH LINKS OBEN-->ZEILE=ZEILE-1
       DEC   @SPPOS        SPALTE=SPALTE-1
       JMP   JOYRTN        ZURUECK
*
MVLTDN INC   @ZEPOS        NACH LINKS UNTEN-->ZEILE=ZEILE+1
       DEC   @SPPOS        SPALTE=SPALTE-1
*
JOYRTN RT                  ZURUECK
*
RTVAL1 LI    R1,8          ZEILE UND SPALTE SIND DER
       JMP   RTVAL3        8.UND DER 9. PARAMETER IM LINKAUFRUF
RTVAL2 LI    R1,9          ZEILE UND SPALTE SIND DER 9.UND DER 10.
```

```
*                                PARAMETER IM LINKAUFRUF
RTVAL3 CLR  RØ
MOV  @ZEPOS,@FAC    ZEILE NACH FPOINTAKKU.
BLWP @XMLLNK        UMWANDLUNG INTEGER-->FPOINT
DATA CIF
BLWP @NUMASG        PARAMETER UEBERGEBEN
INC  R1             NAECHSTER PARAMETER IST SPALTE
MOV  @SPPOS,@FAC    SPALTE NACH FPOINTAKKU.
BLWP @XMLLNK        UMWANDLUNG INTEGER-->FPOINT
DATA CIF
BLWP @NUMASG        PARAMETER UEBERGEBEN
RT                  ZURUECK
```

```
*PROGRMMSEGMENT "@SCRMEMO#
*LAENGE=15 ZEILEN
*
SCRSAV LWPI MYWS              EIGENE ARBEITSREGISTER LADEN
       LI   RØ,>ØØØØ          STARTADRESSE FUER SCREEN IMAGE TABLE
       LI   R1,SCREEN         STARTADRESSE  FUER SCREENBUFFER
       LI   R2,768            768 CHARACTERS WERDEN GELESEN
       BLWP @VMBR
       BL   @BSRTN            --->BASIC
*
SCRRET LI   RØ,>ØØØØ          STARTADRESSE SCREEN IMAGE TABLE
       LI   R1,SCREEN         STARTADRESSE SCREENBUFFER
       LI   R2,768            768 BYTES WIEDER
       BLWP @VMBW             IN DAS VDP-RAM SCHREIBEN
       BL   @BSRTN            --->BASIC
```




```
*PROGRAMMSEGMENT "@CHARSCAN#"
*TABTET DAS ZEICHENFELD AB UND
*LIEFERT DIE STRING VARIABLE
*CHAR# AN DAS BASIC PROGRAMM
*LAENGE=1Ø2 ZEILEN
*
CHRSCN LWPI MYWS              EIGENE ARBEITSREGISTER
*
       LI   RØ,>4ØØØ          ERSTES BYTE IN STRING ENT-
       MOVB RØ,@STRING        HAELT DIE LAENGE.
       LI   R6,STRING+1       R6 IST DER ZEIGER AUF STRING
       CLR  R7                R7 WIRD ALS HILFSREGISTER BENUTZT
       LI   RØ,>ØØ22          ANFANGSADRESSE DES ZEICHENFELDES
       LI   R1,CHRBUF         ADRESSE DES CHARACTERBUFFERS
       LI   R2,4              4 BYTE NACH CHRBUF UEBERTRAGEN
***********************************
*SCNLP1 TASTET DIE LINKE SEITE DES  *
*ZEICHENFELDES ZEILE FUER ZEILE AB  *
***********************************
SCNLP1 INC  R7
       BLWP @VMBR
       BL   @SUBSCN
       MOVB @CHRBYT,*R6+      CHRBYT IN STRING SCHREIBEN
       AI   RØ,>ØØØ4          WIEDER VIER BYTE LESEN
       LI   R1,CHRBUF
       BLWP @VMBR
       BL   @SUBSCN           ZU CHRBUF PABSENDES CHRBYT SUCHEN
       MOVB @CHRBYT,*R6+      CHRBYT IN STRING SCHREIBEN
       AI   RØ,>ØØ1C          NAECHSTE ZEILE
       LI   R1,CHRBUF
       CI   R7,>ØØ1Ø          ALLE ZEILEN GELESEN?
       JNE  SCNLP1            WENN NICHT DANN WEITER
*
       CLR  R7
       LI   RØ,>ØØ2A          VIER BYTE IN
       LI   R1,CHRBUF         CHARBUF EINLESEN
       LI   R2,4
*********************************
*SCNLP2 TASTET DIE RECHTE HAELFTE*
*DES ZEICHENFELDES AB            *
*********************************
SCNLP2 INC  R7
       BLWP @VMBR
       BL   @SUBSCN           VERGLEICHEN MIT HXDATA
       MOVB @CHRBYT,*R6+      CHARBYT IN STRING SCHREIBEN
       AI   RØ,>ØØØ4          NEUE ADRESSE
       LI   R1,CHRBUF         WIEDER VIER BYTE
       BLWP @VMBR             IN CHRBUF EINLESEN
       BL   @SUBSCN           VERGLEICHEN MIT HXDATA
       MOVB @CHRBYT,*R6+      CHRBYT IN STRING SCHREIBEN
       AI   RØ,>ØØ1C          NAECHSTE ZEILE
       LI   R1,CHRBUF
       CI   R7,>ØØ1Ø          ALLE 16 ZEILEN ABGETASTET?
       JNE  SCNLP2            WENN NICHT  DANN WEITER
*
*
       MOVB @HØØ,@STATUB
       CLR  RØ
       LI   R1,1
```

```
       LI   R2,STRING
       BLWP @STRASG
       BL   @BSRTN
***********************************************
*SUBSCAN VERGLEICHT DIE VIER BYTES IN CHRBUF WORT*
*FUER WORT MIT HXDATA.JE EIN HXBYTE ENTSPRICHT   *
*EINEM WORT AUS HEXDTA.                          *
***********************************************
SUBSCN CLR  @CHRBYT            CHRBYT LOESCHEN
       CLR  @FLAGØ1            FLAGØ1  LDESCHEN
       LI   R3,HEXDTA          STARTADRESSE HEXDTA
       LI   R4,CHRBUF          STARTADRESSE CHRBUF
       LI   R5,HXBYTE          STARTADRESSE HXBYTE
       MDVB *R5+,@CHRBYT       ERSTES BYTE AUS HXBYTE
*                              NACH CHRBYT
COMP1  INC  @FLAGØ1            FLAGØ1 =1
       C    *R4+,*R3+          ERSTES WORT CHRBUF=ERSTES
*                              WORT IN HEXDTA?
*
       JNE  NXTWRD             WENN NICHT DANN NAECHSTES WORT
       C    @FLAGØ1,@VALUE2 ZWEITES WORT GEFUNDEN?
       JEQ  SCANRT             JA->ZURUECK
       JMP  CDMP1              ZWEITES WORT VERGLEICHEN
*
NXTWRD
       C    @FLAGØ1,@VALUE1 FLAGØ1=1? EIN WDRT GEFUNDEN
       JEQ  NXWRD1
*
       C    @FLAGØ1,@VALUE2 FLAGØ1=2? BEIDE WDRTE GEFUNDEN
       JEQ  NXWRD2
*
NXWRD1 MDV  *R3+,@WRDBUF       NAECHSTES WORT AUS HXDATA
       LI   R4,CHRBUF          R4  WIEDER AUF ANFANG VDN CHRBUF SETZEN
       MOVB *R5+,@CHRBYT       NAECHSTES HXBYTE NACH CHRBYT
       CLR  @FLAGØ1            FLAGØ1 LDESCHEN
       JMP  COMP1
*
NXWRD2 LI   R4,CHRBUF          R4 WIEDER AUF ANFANG VON CHRBUF SETZEN
       CLR  @FLAGØ1            FLAGØ1 LOESCHEN
       MDVB *R5+,@CHRBYT       NAECHSTES HXBYTE NACH CHRBYT
       JMP  COMP1              WIEDER VERGLEICHEN
*
SCANRT RT                      ZURUECK
```

```
*DAB PROGRAMMSEGMENT "@DRAW#"
*ENTHAELT DIE ZEICHENRDUTINE
*LAENGE=152 ZEILEN
*
DRAW   LWPI MYWS             EIGENE ARBEITSREGISTER
*
       CLR  @TACT            TACT AUF NULL SETZEN
       BL   @DRWVAL          LINKPARAMETER FUER ZEICHENPROGRAMM UEBERNEHMEN
       BL   @SPTSET          SPRITE CURSOR SETZEN >+<
       BL   @VALTST          UEBERPRUEFEN OB ZEPDS UND SPPOS INNERHALB DER
*                            ZULAESSIGEN GRENZEN SIND
       BL   @VALCLC          NEUE POSITION IN VDP-RAM ERRECHNEN
       BL   @SPTLCT          SPRITE CURSDR AUF DIESE POSITION SETZEN
*************************************************
*DRAWL1 UBERPRUEFT OB TACT=TVRG,WENN JA DANN      *
*WERDEN ZEILE UND SPALTE AN DAS BASICPRDGRAMM UEBERGEBEN*
*************************************************
DRAWL1 BL   @DELAY           PAUSE EINSCHALTEN
       INC  @TACT            TACT UM EINS ERHOEHEN
       C    @TACT,@TVRG      TACT MIT TVRG VERGLEICHEN
       JNE  DRAWL3           WENN NOCH NICHT GLEICH DANN WEITER MIT DRAWL3
       BL   @RTVAL2          ZEILE UND SPALTE AN DIE LINKPARAMETER UEBERGEBEN
       BL   @BSRTN           ZURUECK INS BASIC
*
DRAWL2 CLR  @TACT            TACT WIEDER AUF NULL
       BL   @DELAY           PAUSE
DRAWL3 LI   RØ,>Ø1ØØ         LINKE TASTATURSEITE UND DER
       MDVB RØ,@>B374        JOYSTICK 1 WIRD GEPRUEFT.
       BLWP @KSCAN
       C    @JOYY,@HØØ       JOYSTICK BEWEGT ?
       JEQ  DRAWL4           WENN NICHT DANN DRAWL4
*
       BL   @JOYCMP          WOHIN WURDE DER JOYSTICK BEWEGT?
       BL   @VALTST          NOCH IN DEN GEWUENSCHTEN GRENZEN?
       BL   @VALCLC          NEUE POSITIDN ERRRECHNEN
       BL   @CHRSET          AN DER NEUEN PDSITION ENTWEDER EIN
*                            ZEICHEN SETZEN DDER LOESCHEN.
       BL   @SPTLCT          CURSOR >+< AUF NEUE PDSITIDN
       BL   @DELAY           PAUSE
       JMP  DRAWL2           UND WEITER
*
DRAWL4 CB   @KEYBRD,@KEY1S  FEUERTASTE GEDRUECKT?
       JNE  DRAWL1           NEIN
       BL   @CHRSET          SONST WIEDER ZEICHEN SETZEN
       JMP  DRAWL2           UND WEITER
********************************************
*VALTST UEBERPRUEFT,OB DIE NEUE ZEILEN UND  *
*SPALTENPDSITION MIT DEN ANGEGEBENEN GRENZEN*
*UEBEREINSTIMMMT,UND KDRRIGIERT DIE WERTE   *
*ENTSPRECHEND.                              *
********************************************
VALTST C    @ZEPDS,@ZEMIN  UNTERE ZEILENGRENZE ERREICHT?
       JEQ  ZINC            WENN JA DANN KORRIGIEREN
       JMP  ZEMAXC          NEIN,DANN OBERE GRENZE PRUEFEN
ZINC   INC  @ZEPOS          ZEILE =ZEILE+1
       JMP  SPMINC          UNTERE SPALTENGRENZE PRUEFEN
*
ZEMAXC C    @ZEPOS,@ZEMAX  OBERE ZEILENGRENZE ERREICHT?
       JEQ  ZDEC            WENN JA DANN KORRIGIEREN
```

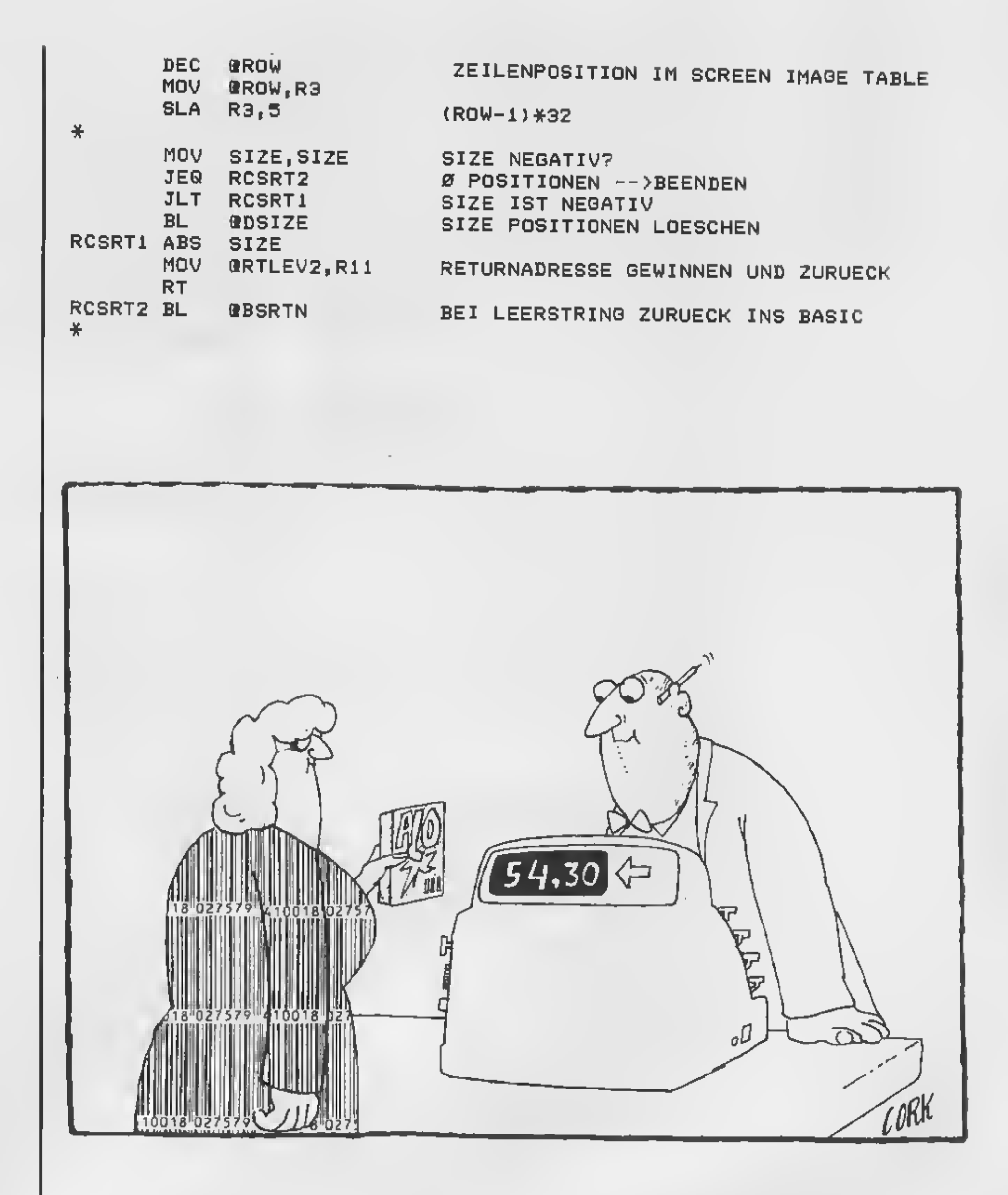
```
       JMP   SPMINC
ZDEC   DEC   @ZEPOS           ZEILE=ZEILE-1
*
SPMINC C     @SPPOS,@SPMIN    UNTERE SPALTENGRENZE ERREICHT?
       JEQ   SINC             WENN JA ,DANN KORRIGIEREN
       JMP   SPMAXC           OBERE GRENZE PRUEFEN
SINC   INC   @SPPOS           SPALTE =SPALTE+1
       JMP   VTSTRT           ZURUECK
*
SPMAXC C     @SPPOS,@SPMAX    OBERE SPALTENGRENZE ERREICHT
       JEQ   SDEC             WENN JA DANN KORREKTUR
       JMP   VTSTRT           ZURUECK
SDEC   DEC   @SPPOS           SPALTE=SPALTE-1
VTSTRT RT                     ZURUECK
***************************
*
CHRSET LIMI Ø
       CB   @KEYBRD,@KEY18    FEUERTASTE GEDRUECKT?
       JEQ  NEWCHR            WENN JA DANN CHARACTER SETZEN
*
DELCHR MOV  @NEWPOS,RØ        NEUE POSITION NACH RØ
       LI    R1,>BØØØ         LEERZEICHEN AUF NEUE POSITION
       BLWP @VSBW
       JMP  CSETRT            ZURUECK
*
NEWCHR MOV  @NEWPOS,RØ        NEUE POSITION NACH RØ
       LI    R1,>DEØØ         TILDE >~< AUF NEUE POSITION SETZEN
       BLWP @VSBW
CSETRT RT                     ZURUECK
*************************
*
SPTSET LI    RØ,>Ø300         SPRITE SETZEN,ENTSPRICHT  DEM CALL
       LI    R1,SPTDTA        SPRITE BEFEHL DES EXTENDED BASIC
       LI    R2,6
       BLWP @VMBW
       RT
************************************
*SPTLCT SETZT DEN SPRITE CURSOR AUF*
*DIE POSITION ZEPOS*B UND SPPOS*B  *
************************************
SPTLCT LIMI Ø
       MOV  @ZEPOS,R1         ZEPOS NACH R1
       DEC   R1               ZEPOS=ZEPOS-1
       LI    RØ,>Ø3ØØ         BEGINN DER SPRITE ATTRIBUTE LIST
       SLA   R1,11            ZEPOS*8
       BLWP @VSBW             IN LISTE EINTRAGEN
       MOV  @SPPOS,R1         SPPOS NACH R1
       DEC   R1               SPPOS=SPPOS-1
       INC   RØ               NAECHSTE EINTRAGSTELLE IM SPR.ATTR.TBL.
       SLA   R1,11            SPPOS*B
       BLWP @VSBW
       RT                     ZURUECK
**************************
DRWVAL MOV   R11,@RTLEV2      RETURNADRESSE SICHERN
       LI    R1,1             ZEMIN UEBERNEHMEN
       MOV   R1,@PRMTR        LINK PARAMETER 1 FUER ZEMIN
       BL   @GETVAL
       DATA 1,24,ZEMIN        GRENZEN 1--24
       DEC  @ZEMIN            GRENZE AUFBEREITEN
```

```
*
       INC  @PRMTR
       BL   @GETVAL
       DATA 1,32,SPMIN
       DEC  @SPMIN
*
       INC  @PRMTR
       BL   @GETVAL
       DATA 1,24,ZEMAX
       INC  @ZEMAX
*
       INC  @PRMTR
       BL   @GETVAL
       DATA 1,32,SPMAX
       INC  @SPMAX
*
       INC  @PRMTR
       BL   @GETVAL
       DATA 1,24,ZEPOS
*
       INC  @PRMTR
       BL   @GETVAL
       DATA 1,32,SPPOS
*
       INC  @PRMTR            UEBERNAHME VON DLYTME
       BL   @GETVAL           DLYTME BESTIMMT DIE LAUF-
       DATA 1Ø,2ØØØ,DLYTME    GESCHWINDIGKEIT DES CURSORS
*
       INC  @PRMTR            UEBERNAHME DER VORGABEZEIT
       BL   @GETVAL           BEI DER EIN RUECKSPRUNG
       DATA 1Ø,1ØØØ,TVRG      IN DAS BASIC ERFOLGT
*
       MOV  @RTLEV2,R11       RETURNADRESSE WIEDERGEWINNEN
       RT                     ZURUECK
       DATA 1,33,SCRCNT
*
       MOV  @RTLEV2,R11
       RT
*
```

```
*PROGRAMMSEGMENT "@DSPACPT#"
*LAENGE=19Ø ZEILEN
*
DBPLAY LWPI MYWS              EIGENE ARBEITSREGISTER LADEN
       BL   @GETRCS           ROW,COL,SIZE UEBERNEHMEN
*
       CLR  RØ
       LI   R1,4              STRING IST DER VIERTE LINKPARAMETER
       LI   R2,DASTRG
       MOVB @BYT28,*R2        STRINGLAENGE=28 BYTES
       BLWP @STRREF           STRING UEBERNEHMEN
       MOVB @DASTRG,LEN       STRINGLAENGE GEWINNEN
       SRL  LEN,8             BYTE-->WORT
       JEQ  DSPRT             LEERSTRING --DANN ZURUECK
*
       LI   R2,DASTRG+1       STRINGADRESSE
DSPYLP MOV  R3,RØ
       A    @COL,RØ           SCREENTABLE OFFSET
       MOVB *R2+,R1           ZU JEDEM ASCII-CODE WIRD 96 ADDIERT
       AB   @BYT96,R1         UM ZUM BASIC PATTERN DESCRIPTOR TABLE
       BLWP @VSBW             OFFSET ZU GELANGEN
*
       DEC  SIZE              ALLE POSITIONEN GEDRUCKT?
       JEQ  DSPRT             JA
       DEC  LEN               STRINGENDE?
       JEQ  DSPRT             JA
       INC  @COL
       C    @COL,@H3Ø         GROESSTE SPALTENPOSITION IST 3Ø
       JLT  DSPYLP
*
DSPRT  B    @BSRTN            BASIC RETURN
*
ACCEPT LWPI MYWS              EIGENE ARBEITSREGISTER LADEN
       LI   RØ,>15ØØ          WERT FUER EINGABEVERZOEGERUNG
       MOV  RØ,@DLYTME        NACH DLYTME UEBERTRAGEN
       BL   @GETRCS           ROW,COL,SIZE FUER ACCEPT UEBERNEHMEN
*
COACPT MOV  @COL,R6
       LI   R7,1
ACPTLP BL   @DELAY            PAUSE FUER EINGABEVERZOEGERUNG
       MOV  R3,RØ             SCREEN TABLE OFFSET
       A    R6,RØ             IN RØ
       BL   @KINPUT           TASTATUR ABFRAGEN
*
       CB   @KEYBRD,@LEFT     TASTE"<--" GEDRUECKT?
       JNE  TRIGHT            NEIN
       C    R6,@COL           LINKES ENDE ERREICHT
       JEQ  ACPTLP            JA DANN HALTEN
       DEC  R6                NEIN ->CURSORPOSITION ERNIEDRIGEN
       DEC  R7
       JMP  ACPTLP
*
TRIGHT CB   @KEYBRD,@RIGHT    TASTE"-->" GEDRUECKT
       JNE  TERASE            NEIN
       C    R6,@H29           RECHTER ANSCHLAG ERREICHT
       JEQ  ACPTLP            JA DANN HALTEN
       C    R7,SIZE           BUCHSTABENZAHL ERREICHT?
       JEQ  ACPTLP            JA POSITION HALTEN
       INC  R6                NEIN POSITION ERNIEDRIGEN
       INC  R7
       JMP  ACPTLP
*
TERASE CB   @KEYBRD,@ERASE    TASTE ERASE  GEDRUECKT
       JNE  TCLEAR            NEIN
       BL   @DSIZE            EINGABEPOSITIONEN WIEDER LOESCHEN
       JMP  COACPT
*
TCLEAR CB   @KEYBRD,@CLEAR    TASTE CLEAR  GEDRUECKT
       JNE  TENTER            NEIN
       B    @BADARG           JA DANN  BAD ARGUMENT ANZEIGEN UND STOP
*
TENTER CB   @KEYBRD,@ENTER    TASTE ENTER GEDRUECKT
       JEQ  GETSTR            JA STRING UEBERNEHMEN
*
       AB   @BYT96,@KEYBRD    SCREENCHARACTER FUER BASIC
       MOVB @KEYBRD,R1
       BLWP @VSBW             CHARACTER ANZEIGEN
*
       C    R6,@H29           RECHTER RAND ERREICHT
       JEQ  ACPTLP            JA
       C    R7,SIZE           CHARACTERANZAHL ERREICHT
       JEQ  ACPTLP            JA
       INC  R6                NAECHSTE SPALTE
       INC  R7                NAECHSTE ZEILE
       JMP  ACPTLP
*
*CHARACTER AUS SCREEN UEBERNEHMEN
*
GETSTR LI   R2,DASTRG         ADRESSE DES STRINGS
       MOVB @HØØ,*R2+         CHARACTERZAHL=Ø
GSTRLP MOV  R3,RØ             SCREEN TABLE OFFSET BERECHNEN
       A    @COL,RØ
       BLWP @VSBR             CHARACTER LESEN
       AB   @BYTM96,R1        NACH ASCII TRANSFORMIEREN
       MOVB R1,*R2+           IN STRING EINLESEN
       AB   @BYT1,@DASTRG     CHARACTERZAHL ERHOEHEN
       DEC  SIZE              NAECHSTES BYTE
       JEQ  ACPTRT
       INC  @COL              NAECHSTE SPALTE
       C    @COL,@H3Ø         LETZTE MOEGLICHE SPALTE ERREICHT
       JLT  GSTRLP
*
ACPTRT MOVB @HØØ,@STATUS      STRING UEBERGEBEN
       CLR  RØ
       LI   R1,4              STRING IST 4.PARAMETER IN LINK
       LI   R2,DASTRG
       BLWP @STRASG           STRING ZUWEISEN AN LINK PARAMETER
*
       BL   @BSRTN            ZURUECK  ZUM BASICPROGRAMM
******************
*TASTATUR ABFRAGEN*
******************
KINPUT MOV  R11,@RTLEV1       RETURNADRESSE SICHERN
       MOVB @HØØ,@>8374
       BLWP @VSBR             MOMENTANEN SCREEN CHARACTER LESEN
       MOVB R1,@SCRCHR
*
KEYLP1 LI   R15,3ØØ           ZUERST CURSOR ANZEIGEN
```

```
       MOVB @CRSR,R1
       BLWP @VSBW
KEYLP2 BLWP @KSCAN           TASTATUR ABFRAGEN
       CB   @KEYBRD,@HEXFF   KEINE TASTE GEDRUECKT
       JNE  KEYRT            WENN DOCH DANN ZURUECK
       DEC  R15              WIEDER ABFRAGEN
       JOC  KEYLP2
*
       LI   R15,35Ø          ZWEITE SCHLEIFE
       MOVB @SCRCHR,R1       VOHER GELESENEN CHARACTER ANZEIGEN
       BLWP @VSBW
KEYLP3 BLWP @KSCAN           TASTATUR ABFRAGEN
       CB   @KEYBRD,@HEXFF   KEINE TASTE GEDRUECKT?
       JNE  KEYRT            WENN JA DANN ZURUECK
       DEC  R15
       JOC  KEYLP3
       JMP  KEYLP1           NOCH EINMAL DAS GANZE
*
KEYRT  MOVB @SCRCHR,R1       SCREENCHARACTER WIEDER ANZEIGEN
       BLWP @VSBW            DADURCH BLEIBT DER CURSOR NICHT
       MOV  @RTLEV1,R11      HAENGEN WENN ENTER GEDRUECKT WIRD
       RT
*
*
DSIZE  MOV  R11,@RTLEV3      RETURNADRESSE GEWINNEN
       MOV  @COL,R4
       MOV  SIZE,R5
       LI   R1,>8ØØØ         LEERZEICHEN ASCII 32
DSZELP MOV  R3,RØ            SCREEN TABLE OFFSET BERECHNEN
       A    R4,RØ            UND BYTE FUER
       BLWP @VSBW            BYTE LOESCHEN
       DEC  R5
       JEQ  DSZERT
       INC  R4
       C    R4,@H3Ø          LETZTE SPALTENPOSITION ERREICHT
       JLT  DSZELP           NEIN DANN WEITER
*
DSZERT MOV  @RTLEV3,R11      RETURNADRESSE GEWINNEN UND
       RT                    ZURUECK
*
GETRCS MOV  R11,@RTLEV2      RETURNADRESSE SICHERN
*                            ROW,COL,SIZE UEBERNEHMEN
SIZE   EQU  9
LEN    EQU  1Ø
*
       LI   R1,1             1.PARAMETER IST ROW(ZEILE)
       MOV  R1,@PRMTR
       BL   @GETVAL
       DATA 1,24,ROW
*
       INC  @PRMTR           2.PARAMETER IST COL(SPALTE)
       BL   @GETVAL
       DATA 1,28,COL
       INC  @COL
*
       INC  @PRMTR           3.PARAMETER IST SIZE(GROESSE)
       BL   @GETVAL
       DATA -28,28,MYWS+18    ZIEL IST SIZE ALSO R9=MYWS+18
*
```

```
       DEC  @ROW              ZEILENPOSITION IM SCREEN IMAGE TABLE
       MOV  @ROW,R3
       SLA  R3,5             (ROW-1)*32
*
       MOV  SIZE,SIZE        SIZE NEGATIV?
       JEQ  RCSRT2           Ø POSITIONEN -->BEENDEN
       JLT  RCSRT1           SIZE IST NEGATIV
       BL   @DSIZE           SIZE POSITIONEN LOESCHEN
RCSRT1 ABS  SIZE
       MOV  @RTLEV2,R11      RETURNADRESSE GEWINNEN UND ZURUECK
       RT
RCSRT2 BL   @BSRTN           BEI LEERSTRING ZURUECK INS BASIC
*
```



```
100 DIM TRICK$(100),HEX$(16)
110 CALL INIT
120 CALL LOAD("DSK1.BSCSUP",
"DSK1.CCAD/OBJKT")
130 CALL SCREEN(13)
140 CALL CLEAR
150 RESTORE 2300
160 GOSUB 650
170 RESTORE 2340
180 GOSUB 710
190 GOSUB 770
200 GOSUB 1120
210 GOSUB 1240
220 GOSUB 540
230 CALL LINK("DRAW",2,3,17,
18,Z1,S1,1000,60,Z11,S11)
240 GOSUB 620
250 GOSUB 560
260 CALL VCHAR(2,20,32,19)
270 CALL LINK("SELECT",2,20,
13,20,2,21,1500,Z2,S2)
280 CALL HCHAR(Z2,20,30)
290 ON Z2-1 GOTO 230,300,360
,360,360,360,360,360,360,360
,360,360,360
300 RESTORE 23B0
310 FOR I=1 TO 2
320 READ SCZ1,SCS1,SCZ2,SCS2
,RTNG,SCOUNT
330 GOSUB 1060
340 NEXT I
350 STOP
360 ON Z2-3 GOSUB 420,390,86
0,880,930,930,930,930,2100,1
300
370 GOSUB 560
380 GOTO 260
390 GOSUB 440
400 GOSUB 570
410 RETURN
420 CALL LINK("INVERT")
430 RETURN
440 CALL LINK("CHRSCN",A$)
450 IF TCT>=100 THEN 460 ELS
E 520
460 FOR I=1 TO 4
470 CALL LINK("DSPLAY",21,1,
1B,"BILDSPEICHER VOLL!")
480 GOSUB 1090
490 CALL HCHAR(21,3,32,18)
500 NEXT I
510 RETURN
520 TCT=TCT+1
530 TRICK$(TCT)=A$
540 CALL LINK("DSPLAY",22,21
,3,STR$(TCT))
550 RETURN
560 CALL LINK("CHRSCN",A$)
570 CALL CHAR(156,SEG$(A$,1,
16))
580 CALL CHAR(157,SEG$(A$,17
,32))
590 CALL CHAR(158,SEG$(A$,33
,48))
600 CALL CHAR(159,SEG$(A$,49
,64))
610 RETURN
620 Z1=Z11
630 S1=S11
640 RETURN
650 READ Z1,S1,A
660 FOR I=1 TO A
670 READ ZE,SP,VAR$
680 CALL LINK("DSPLAY",ZE,SP
,LEN(VAR$),VAR$)
690 NEXT I
700 RETURN
710 READ A
720 FOR I=1 TO A
730 READ NR,CR$
740 CALL CHAR(NR,CR$)
750 NEXT I
760 RETURN
770 CALL HCHAR(1,3,60,16)
780 CALL VCHAR(2,2,62,16)
790 CALL VCHAR(2,19,62,16)
800 CALL HCHAR(18,3,60,16)
810 CALL HCHAR(18,22,156)
820 CALL HCHAR(19,22,157)
830 CALL HCHAR(18,23,158)
840 CALL HCHAR(19,23,159)
850 RETURN
860 LC=126
870 GOTO B90
880 LC=32
890 FOR I=2 TO 17
900 CALL HCHAR(I,3,LC,16)
910 NEXT I
920 RETURN
930 RTNG=Z2-7
940 CALL LINK("DSPLAY",20,1,
28,"SCROLL SEKTOR WAEHLEN!")
950 SCOUNT=1
960 ON RTNG GOTO 970,970,102
0,1020
970 CALL LINK("SELECT",2,3,1
6,18,2,3,1000,SCZ1,SCS1)
980 GOSUB 1090
990 CALL LINK("SELECT",SCZ1+
1,SCS1,17,1B,17,18,1000,SCZ2
,SCS2)
1000 GOSUB 1090
1010 GOTO 1060
1020 CALL LINK("SELECT",2,3,
17,17,2,3,1000,SCZ1,SCS1)
1030 GOSUB 1090
1040 CALL LINK("SELECT",SCZ1
,SCS1+1,17,18,17,18,1000,SCZ
2,8C82)
1050 GOSUB 1090
1060 CALL LINK("SCROLL",SCZ1
,SCS1,SCZ2,SCS2,RTNG,SCOUNT)
1070 CALL HCHAR(20,1,32,32)
1080 RETURN
1090 CALL SOUND(200,444,0)
1100 CALL SOUND(1,30000,0)
1110 RETURN
1120 RESTORE 2390
1130 FOR I=1 TO 16
1140 READ HEX$(I)
1150 NEXT I
1160 READ CHAR$
1170 POS$="0123456789ABCDEF"
1180 FOR I=1 TO 32 STEP 2
1190 VAR$=HEX$(POS(POS$,SEG$
(CHAR$,I,1),1))&HEX$(POS(POS
$,SEG$(CHAR$,I+1,1),1))
1200 VAR$=VAR$&HEX$(POS(POS$
,SEG$(CHAR$,I+32,1),1))&HEX$
(POS(POS$,SEG$(CHAR$,I+33,1)
,1))
1210 CALL LINK("DSPLAY",INT(
I/2)+2,1,16,VAR$)
1220 NEXT I
1230 RETURN
1240 GOSUB 560
1250 FOR I=1 TO 4
1260 CALL LINK("SCROLL",2,3,
17,18,I,4*I)
1270 NEXT I
1280 GOSUB 560
1290 RETURN
1300 CALL LINK("SCRSAV")
1310 CALL POKEV(768,00,00,00
,00)
1320 IF TCT=0 THEN 1330 ELSE
 1380
1330 CALL CLEAR
1340 CALL LINK("DSPLAY",10,1
,28,"BILDSPEICHER IST LEER!!
!!!!")
1350 CALL SOUND(3000,196,0,2
62,0,330,0)
1360 CALL SOUND(1,30000,30)
1370 GOTO 1460
1380 RESTORE 2360
1390 READ A
1400 CALL CLEAR
1410 GOSUB 560
1420 GOSUB 660
1430 GOSUB B10
1440 CALL LINK("SELECT",4,6,
9,6,3,6,1500,Z2,S2)
1450 ON Z2-3 GOTO 1490,1720,
1460,1780,1970,2050
1460 CALL LINK("SCRRET")
1470 GOSUB 540
1480 RETURN
1490 IF TCT=0 THEN 1440
1500 RESTORE 2410
1510 READ A
1520 GOSUB 660
1530 TCT1=0
1540 FOR BILD=1 TO TCT
1550 A$=TRICK$(BILD)
1560 CALL LINK("DSPLAY",20,1
1,3,STR$(BILD))
1570 GOSUB 570
1580 CALL SOUND(10,1000,0)
1590 CALL LINK("SELECT",22,4
,24,4,22,4,1500,Z3,S3)
1600 ON Z3-21 GOTO 1630,1610
,1640
1610 TRICK$(BILD)=""
1620 TCT1=TCT1+1
1630 NEXT BILD
1640 FOR I=1 TO TCT
1650 IF TRICK$(I)="" THEN 16
60 ELSE 1680
1660 TRICK$(I)=TRICK$(I+1)
1670 TRICK$(I+1)=""
1680 NEXT I
1690 TCT=TCT-TCT1
1700 CALL HCHAR(20,1,32,165)
1710 GOTO 1440
1720 FOR BILD=1 TO TCT
1730 A$=TRICK$(BILD)
1740 IF A$="" THEN 1760
1750 GOSUB 570
1760 NEXT BILD
1770 GOTO 1440
1780 DELETE "DSK1.CHARCODE"
1790 OPEN #1:"DSK1.CHARCODE"
,DISPLAY ,VARIABLE B0
1800 FOR CN=1 TO TCT
1810 TEIL$=TRICK$(CN)
1820 FOR TEIL=1 TO 49 STEP 1
6
1830 VA$(INT(TEIL/16)+1)=SEG
$(TEIL$,TEIL,16)
1B40 NEXT TEIL
1850 FOR D=1 TO 4
1860 DRUCK$=DRUCK$&"        D
ATA "
1870 FOR I=1 TO 16 STEP 4
1880 DRUCK$=DRUCK$&">"&SEG$(
VA$(D),I,4)&","
1890 NEXT I
1900 DRUCK$=SEG$(DRUCK$,1,LE
N(DRUCK$)-1)
1910 PRINT #1:DRUCK$
1920 DRUCK$=""
1930 NEXT D
1940 NEXT CN
1950 CLOSE #1
1960 GOTO 1440
```

BÖRSE

TI 99/4A
Wer hat noch Sportspiele? Bitte kurze Spielbeschreibung mit Liste der benötigten Geräte an
K. Wolters, Corneliastr. 9, 43 Essen 1, schicken!!!!

Dt. Handbuch für E/A zu verkaufen. W. Götz, 0711/864053

Suche: externe ansteckbare 32 k-RAM, Sprachsynthesizer und Buch Rogers Modul, K. Wolters' 0201/772141

Suche:
ext. 32 K-RAM Erweiterung, Buck Rogers Modul und Ex.-Basic (Mechatronic)
Zu melden bei K. Wolters, Tel. 0201/772141

Verkaufe: Schachmodul und TI-Extended Basic. Zu melden bei
K. Wolters, Telefon: 0201/772141

Suche dringend Kons. mit Rek. Kabel und Pal. Mod. bis DM 100. Tel. 3137650 (089) ab 19 Uhr)

Suche günst. Drucker mit V24-Anschl. incl. Kabel. Tel. 089/3137650 ab 19.00 Uhr

---

Günstig abzugeben!!! Ex-B. dt. Nachbau + 32 K + Cent. + Soft w. 500 DM.
Tel. 02955/6546

Zu verk. orig. Ti-Ext. Basic Modul u. Lit., Spiele in Ext. Basic VB 190,– DM Tel. 06441/52556

Suche original TI-Joysticks bis 30 DM
Tel. (030) 7865425

Verkaufe Masch.Progr. (ED/Ass) auf Kassette) nur EX-Basic und 32 K. nötig. – Infos gegen Rückporto –
B. Dusny, Mülldorfer Str. 31, 5205 St. Augsutin I

Verkaufe:

Speech Synthesizer	120,–
Speech Editor	30,–
Terminal Emulator II	60,–
Mini Memoriy	160,–
Dateiverwaltung Modul	50,–
Invaders	25,–
Munchman	25,–
Parsec	30,–

Michael Strub, Hauptstraße 8774 Leuggelbach (CH)

****ACHTUNG****
Suche TI-User zwecks Spiele-Austausch (in Ex-Basci)
Bitte melden bei: Olav Keith, Tel. 0221/373530 *ab 19 h*

Tausche Prg. in TI+Ext. Basci! Liste an: H.-J. Eckers, Hermann-Hesse-Str. 7, 4050 Mönchengladb. 1, Interessenten für Clubbeitritt („TECHNOMANIA" aus MG) möglichst aus Raum D, MG, NE, KR; bitte ebenfalls bei mir schriftlich melden!!!

Wer verkauft günstig Schach-Modul, Sprachsynthesizer od. 32 k-Erweiterung? Angebote an: H.-J. Eckers, Hermann-Hesse-Str. 7, 4050 Mönchengladbach I, Tel. 02161/52603

Verkaufe: Komplettes TI-99 System! Weitere Auskünfte: Oliver Siffrin, Tel. 06821/7522

```
1970 DELETE "DSK1.CHARMEM"
1980 OPEN #1:"DSK1.CHARMEM",
VARIABLE 163
1990 FOR CN=1 TO TCT
2000 PRINT #1:CHR$(0)&CHR$(C
N)&CHR$(147)&CHR$(200)&CHR$(
64)&TRICK$(CN)&CHR$(0)
2010 NEXT CN
2020 PRINT #1:CHR$(255)&CHR$
(255)
2030 CLOSE #1
2040 GOTO 1440
2050 FOR I=1 TO TCT
2060 TRICK$(I)=""
2070 NEXT I
2080 TCT=0
2090 GOTO 1440
2100 GOSUB 880
2110 CALL LINK("DSPLAY",2,1,
16,"BITTE CHAR/CODE")
2120 CALL LINK("DSPLAY",3,1,
16,"EINGEBEN!0-9/A-F")
2130 CHAR$=""
2140 FOR I=1 TO 4
2150 CALL LINK("ACCEPT",I+4,
1,-16,VAR$)
2160 FOR VCT=1 TO 16
2170 IF POS(POS$,SEG$(VAR$,V
CT,1),1)=0 THEN 2200
2180 NEXT VCT
2190 GOTO 2250
2200 CALL LINK("DSPLAY",16,1
,16,"EINGABEFEHLER!!!")
2210 CALL SOUND(500,444,0)
2220 CALL SOUND(1,30000,30)
2230 CALL HCHAR(16,3,32,16)
2240 GOTO 2150
2250 CHAR$=CHAR$&VAR$
2260 NEXT I
2270 GOSUB 880
2280 GOSUB 1180
2290 RETURN
2300 DATA 2,3,15,24,1,COMPUT
ER AIDED CHAR DESIGN,2,19,ZE
ICHNEN,3,19,STOP,4,19,INVERT
,5,19,CHARSAVE
2310 DATA 6,19,FUELLEN,7,19,
LOESCHEN
2320 DATA 8,19,SCRLDOWN,9,19
,SCRLUP,10,19,SCRLRIGHT,11,1
9,SCRLLEFT,12,19,C-EINGABE
2330 DATA 13,19,TRICKFILM,14
,19,&SPEICHERN,22,1,GESPEICH
ERTE BILDER=0
2340 DATA 7,156,000000000000
0000,157,0000000000000000,15
8,0000000000000000,159,00000
00000000000
2350 DATA 126,7E8199A5A59981
7E,60,00003C24243C,62,00003C
24243C0000
2360 DATA 7,1,1,--BILDSPEICH
ER BEARBEITEN---,4,5,BILDAUS
WAHL,5,5,FILM AB,6,5,ZURUECK
2370 DATA 7,5,SPEICHERN MERG
E FORMAT,8,5,SPEICHERN E/A F
ORMAT,9,5,SPEICHER LOESCHEN
2380 DATA 1,1,24,16,1,24,1,1
6,24,32,2,24
2390 DATA "     ","   ~","  ~
 "," ~~"," ~  "," ~ ~"," ~~
 "," ~~~","~   ","~  ~","~ ~
 ","~ ~~","~~  "

2400 DATA "~~ ~","~~~ ","~~~~
,00F0484848484BF0000F08080E0
8080F88D8A88888888800000F090
90909090F
2410 DATA 4,20,3,BILD NR.,22
,3,BILD OK,23,3,BILD LOESCHE
N,24,3,ZURUECK
```

BÖRSE

Graphik-Tabl. 150 DM, Flugsim (Cas) 35 DM, TI-Modul American Football Sup Graphik 65 DM Kopf 095116846

TI-Tunnel's of Doon dt. Übersetzung für 15,– DM Vorkasse. M. Redlich, Eichenweg 3, 4620 Castrop-Rauxel 2

Suche preisgünstig TI-Writer, Multiplan
Dieter Redlich, Eichenweg 3, 4620 Castrop-Rauxel 2

Suche Anwenderprogramme. Verkaufe etliche Module + Erweiterungen. Horst Nietowski, 02173/15395 Langenfeld

---

Verkaufe: original TI-EX-Basic. Tel. 040/3193417 Anruf nach 18 h

---

Ext.-Basic + XB-Lehrg. + 50 XB-Prgrm. 250 DM, 10 Module (Schach, M.-Maker usw.) ab 15 DM, TI-Bücher ab 9 DM, Telefon 02174/40654

TI-99A, RS232 ext. 32k, P-Box incl. Disk, etl. Module, einzeln zu v.
K. Berdon, Odenwaldstr. 12, 6056 Heusenstamm
Tel. 0610462185

Verkaufe Ext.-Basic für VB 160,– DM + Datenverwaltung & Analyse VB 45,– DM + Othello VB 20,– + Parsec VB 40,– DM + Konsole (etwas defekt) VB 60,– DM!
Matthias Orf, Birkenallee 34, 3507 Baunatal 1,
Tel. 0561/497990

TI-CLUB BAUNATAL bietet: Clubheft mit 25 Seiten, Prg.-Speicher (400 Programme!), Entfernen von Listschützen, Drucker, toller Ausweis für 2,– DM pro Monat!! Info gegen 50 Pf. oder aktuelles Clubheft gegen 3,– DM anfordern bei: TCB, Matthias Orf, Birkenallee 34, D-3507 Baunatal 1, Tel. (0561) 497990

TI-CLUB BAUNATAL sucht Kontakt zu anderen Ti-User-Clubs zwecks Info- und Gedankenaustausch.
TI-CLUB BAUNATAL, Matthias Orf, Birkenallee 34, D-3507 Baunatal 1,
Tel. 0561/497990

TI-32K Speichererweiterung f. Modul Box zu verkaufen 300 DM 06441/74830

HALLO 99' User! Wie, Sie sind noch nicht im MONSTERVISION Club! Dort gibt es monatlich ein 32seitiges Magazin mit vielen Vorteilen für alle Mitglieder. Eine Clubgebühr ist nicht vorhanden! Gratis-Infos bei: MV Club, Gratis-Info. Uesener Ring 30, 2807 Achim

---

TI99/4A + Box + Disk + Ext Basic + Sp. Synth + Chess + Invaders + Joysticks + Literatur DM 1100.
Tel. (SA, SO) 09621/85143

Verk. TI+Rec Kabel + Rekorder + Joaystick + 5 Module + Basic Lehrgang + Literatur für 340 DM.
Tel. 06721/43307

Verkaufe: Ti + Ext. Basic + Parsec + Music-Maker + viel Software für nur 250,– DM Ideal für Einsteiger
Markus Schröpfer, Schulstr. 22, 8567 Neunkirchen

Suche TI-User Club in Berlin (West) zwecks Informations-Austausch.
Ab 19 h. Tel. 811 51 46

---

6 Top-Hits direkt vom Autor. Z.B. Nanuk der Eskimo, Stardust, Alien-Landing. Senden Sie 30,– DM im Kuvert an: W. Döltsch A.D. Hinterstein 10 6108 Weiterstadt 3

Letzte Lösung! Brauche dringend Geld. Verk. TI99/4A + TI XBasic + Bas. / XBas. Lehrpr. + Lehrbücher / Lit. + Joy Adapter + 2fach Rec. Kabel + Chis. Trail + ca. 280 Progr. Alles gepfl. VB 450 DM. Tel. 0231/875916 ab 18 h

---

Wegen Syst.auflösung zu verk. Org. TI RS232/V 24 100,–, Speechs. 70,–, Module von 20,– bis 50,–, Compiler 100,–, Atari Module Jungle, Dig, Kong je 40,– Tel. 09151-95153

Suche Heft Computer Praxis 12/84 in einigermaßen gutem Zustand. Tausche gegen 100 EXB (Tib) Prg's oder zahle bis zu 5 DM. Verkaufe engl. Handbuch für das EXBModul für 15 DM.

Tausche Programme in TI und Ex-Basic. Habt Ihr Lust? Meldet Euch einfach bei mir: 07156/34 941 od. schriftl. Carlos Jarque, Ludwigsburger Str. 14 / 7257 Ditzingen 1

ZU VERSCHENKEN habe ich nichts ... aber da ich mir eine 32K Erweiterung kaufen möchte, bin ich gezwungen, etwas aus meiner Computersammlung zu verkaufen. Z.B. 11 Adventure Cass. original TI Stück 15,– DM., M*A*S*H* 40,– DM, TI Bücher 20–30 DM, Computer hefte z.B. HC, Compu-tronic u.s.w. 2,– 50 – 3,– DM, Spielmodule für ATARI Compuer z.B. GYRUSS, STAR-TRECK, Popeye, O-BERT u.s.w. 30–50 i DM. Info gegen frankierten Rückumschlag bei: Manfred Lipowski, In der Wanne 165, 462 Castrop-Rauxel 4, Tel. 02305/72237

Komplett TI99/4A Konsole + PBox + RS232 + Ext. Basic + Laufwerk + 32 Kram + Minimemory + Schach + Lit. + Softw.
Tel. 07231/41436

Hey TI-Freaks! Wer hat Lust mit Nappsoft Programme in TI + Ex. Basic zu tauschen? Liste an Martin Roth, Bruckner Str. 3, 6680 Neunkirchen 7

Verkaufe TI99/4A+dt.X-Basic+orig. TI-Box m. Controller+Laufwerk+Spielmodule+Joystickadapter+ viel Software auf Disk. wegen Systemwechsel. Abgabe gegen Höchstgebot! C. Reusch, Tel. 0241/172129

Hallo TI E/A User. Wer von Euch hat Lust, mit mir Prg's zu tauschen. Markus Junghans. Tel. 06106/74182

Verkaufe TI99/4A + EX + 14 Module + Literatur + Zubehör. Alles original verpackt. Tel. 02174/40616 nach 15 Uhr

---

Schüler sucht für TI99/4A 32 K Ext und Ext. Basic o. Exb. II + M. Stief/Sandstücke 21/28 Bremen 61

Achtung TI-Adv. Freunde!!! Jetzt gibt's DAS Graphic-Adv., auf das Ihr schon lange gewartet habt. (ca. 16k) TI-Basic; kein Adv.-modul notwendig. Für DM 5 + Vers. Bei J. Laux, Schulgartenstr. 20, 6638 Dillingen. Es lohnt sich

---

Verk. Ex. Basic + Handbuch + 100 Super Programmen für nur 200 DM. Ralf Ludwig, Am Mergelsberg 31, 4000 Düsseldorf 12, Tel. 0211/297042

994A+Ex.B. (engl. u. Deu.H. buch) + Kass. rec. + Kabel + Spech-Syn. + Munchman+ Engl. Grammatik+Literatur 600,–. Tel. 069/554026 T. Veith

---

Tausch*Musik Maker* Datenverw. u. Analyse* Comp. Kurs Bis Nr. 56* U.A.M. * Su. Extern Erw. * Assembler * U.A.Ang.*

Schnittstelle für Direktanschluß eines Druckers sowie viele tolle Module-Preis VB Tel. 06103/72518

Verkaufe meine Spielesammlung (ca. 300–350 Prg's; 70 % EXB, 30 % Tib) an den am meist bietenden!!! Übernehme Porto + Cassette N (15 x C 60)! Tel. 07156/34941 oder Carlos Jarque, Ludwigsburgerstr. 14, 7257 Ditzingen 1

Verkaufe 99/4A mit Box, Dsk, 32K, X-Basic, E/A, dt. E/A Kurs, Statistik, Modulexpander und viel Literatur. Nur komplett! Neu über 3000,– jetzt 1600,– 05300/485 Uli

---

Verkaufe ExBasic + Adventure Modul + 12 Cassetten +Literatur für 250 DM Tel. 08638/67495 ab 19 h

---

Verk. Assemblerkurs: Asem-4 Band I + 2 + Disk DM 50,– C. Kater 7, Rue de Schoenfels, L-7432 Gosseldange Tel. 328060

Suche HARDCOPY-Programm für EPSON-Drucker. Alexander Rupp, Kalmanstr. 45, 6600 Saarbrücken; Tel.: 0681/45134

Suche Schaltungsunterlagen für TI-Druckerinterface sowie Schnittstelle RS232. R. Schinkel, Hamburger Str. 20717, 2200 Elmshorn

---

Verk. TI99/4a-Konsole + Netzteil + 32 K extern incl. durchgef. Bus u. Centronic + Ext-Basic Modul + Rec. Kabel + Centronic-Kabel + Joyst.-Adapter + Literatur einzeln od. zusammen, Preis: VHS
Tel. 05283/1850 Mo–Mi 18 h

Verk. TI und Ex Basic Programme. Info gegen Rückporto. B. Knedel, Tulpengasse 16, 3171 Weyhausen, T. 05362/71187

BÖRSE

Verkaufe kompl. für 400 DM TI99/4A anschlußfertig + 1 Modul Ex.-Ba. mit Handbuch + Cas.-Rec. M. Kabel, 3 Bücher, 6xTI-Journal + 8xTI-Revue + I Monitorkabel. Kl. Entinger, 6650 Homburg, Westring 22, Tel. 06841/71693 nach 17 h

---

Verkaufe 20 TI-Progr. Stck./ 8,— DM wie z.B. Flugsimulator, Star-Strike, Uambler etc. Ruft an: 06874/6705

TI99/4A Speechsynt. 2 Joyst. 12 Module (ext. Bas., Parsec, Schach ...) Kassetten, viel Literatur 500,– DM Tel. 06834/41660 (Klose)

Verkaufe TI-Writer + Multiplan zus. DM 250,—. Tel. 04262/1205

TI99/4A; neuwertig; Handbuch, orig. Verp. + Recorderkabel: 170 DM. B. Dobrick, Hohenstaufenstr. 17, 7340 Geislingen

---

Wer im Raum Hi/H/BS hat CPS 99 Atronic. Bitte melden. Tel.: 05069/6538. Rufe zurück. JORK Warnecke

---

Suche Bedienungsanleitung zur Diskvers. des X-Basic II. Wer kann helfen? Tel. 02051 66950 ab 20 Uhr.

TJ Expannsion Box ller mit Netzteil DM I20 Diskgehäuse mit Kabel + Netzt. DM 50 zu verk. 08233/6653

Verk. (orig. TI) Diagnostic-Mod + RGB /Modulator (H4/85) Stulen Roerdomps 7671WL Vriezenveen, Tel. 0933549962650

Suche PBox + Disklw (ev. nicht Org. TI) + 32K + RS 232 + evTI Writer + Speech-Synth + mit Handbüchern Tel. 0711 843775

Verk. orig. Ex-Basic, unbenutzt. suche Speech-Synth. + TE 2, Frank Brengel, 089/1231332 ab 18 Uhr

---

Suche PAL Modulator PHA 2036 for TI-99A Channel 36 auch Spielprogramme.

Suche TIBox mit Contr/Ext. DSKContr/Sprachsynthersizer/Avend u. ExModul. Dieter Wagner, 7750 Konstanz, Leipzigerstr. 9

Verkaufe Module: Speech Editor, Attack, Ext. Basic, etc. Tel. 04131-55457 nach 19 Uhr

---

Tippe Listings ab. 10 DM (inc. Postg.) in Umschlag mit Listing abschicken. Schicke alles + Kass. zurück an: Aronica Luigi, Deeler Weg 14, 5000 Köln 71

Verkaufe wegen Systemerweiterung 3 original TI-Laufwerke, einen Diskcontroller und eine 32-KByte-Speichererweiterung. Liebald Heiko, Eichenweg 7, 7914 Pfaffenhofen o.d. Roth

---

Viel Zubehör für TI abzugeben wegen Systemaufgabe. Z.B.: Printer/Plotter, externe Cent. Schnittstelle (mit Kabel und durchge. Bus), original Joystick Ext. Basic, alle Adventure; Mini Assembler, Schachmeister; Moon Patrol und vieles mehr. Informationen bei Thorsten Rauer, Beetstr. 64, 4902 Bad Salzuflen, 05222/ 13182

Assembler-Programme f. E/A, XB, MM + 32 K: GPL-Disassembler; FAST-COPY (Sektorkopierer) kopiert jede Disk in drei Durchgängen. Info gg. Rückumschlag. Alles sofort lieferbar! M. Eichhorn, Ziegelheck 1, 6240 Königstein 4

Verkaufe: TI99/4A Konsole + Extended-Basic II plus 2 Module (Microsurgeon und Schachmeister) + Recorderkabel + Joystickadapter + 2 Joysticks + Prgm's + Hefte VB: DM 500,– ÖS 3500,– T.: Österreich / 0662/26671 (Salzburg)

Suche ext. Laufwerk + Disc-Kontroller günstig. Tel. 06805/8393

Verk. Minimem m. dt. Handbuch 190,— Pers. Rep. gen. Othello, Parsec, Stat., Ger., Householdbudg. Man. je 20,— J. Kupzig 02208/4165

STOP! Habt ihr ein TI zuhause stehen? Habt Ihr Lust, mit mir PGM's zu tauschen? Oder wollte Ihr sie kaufen? Oder sucht Ihr andere TI-User? Dann schreibt mir! M. Kugelmann, Nassauerstr. 4, 6272 Niederhausen!

Einmalige Chance! Junger TI-Freak möchte mit dir PGM's tauschen! Mords Software in TI, Ex-Basic oder Exbasic + RAM-Pack! Stelle auf Wunsch auch PGM's her! Adr.: M. Kugelmann, Nassauerstr. 4, 6272 Niederhausen!

Verkaufe TI99/4A + P-Box + 32K + Controller + Disklw. + Ex. Centrois + Joystik + EIA + d.t. Handb. + Ex-Basic + dt. Hb. + 6 Sp. Module + dv. TI Spezialb. + Super Grafic + d.v. Programme VB 1500,—. Jan Böhme, Röweland 16, 2000 HH 62

Verk. TI99/4A, P-Box, 32 K Erw., RS 232, Disk-Contr., Disk-Laufwerk, Sprachsyn., Ex.-Basic, Editor/Assembler, Mini-Memory, div. Module, TOP-Software (M.-Code), Literatur usw. VB 2000,— DM. Tel. 07132/37608

Verk. TI-99/4A + Joystick + Datasette + ca. 120 Programme + 6 Module + Fachliteratur. VB 280,– DM. M. Schreiner. Tel. 06103/ 81815

Verk. Disk-Contr., Laufwerk—orig. TI für P-Box. Suche Peri-Box orig. TI. Angebote an: Richert, 02921/2626

Verkaufe TI 99/4A + X-Basic + P-Box m. Laufwerk + Schnittstellenkarte + Datenverw. + Statistik + Recorder m. Kabel + Literatur für DM 1500 VB. Telefon: 06403/71104

NEU: Editor / Assembler auf Kassette: Nur 70 DM! Super schnell und komfortabel! Tel. 0561/887129: Lothar Krauß

Verkaufe TI/Ex-Basic für Anfänger und Fortgeschrittene. Tel. 0211/422216

Verkaufe original TI-Ex-Basic Handbuch (in englisch) Tel. 0211/422216

Achtung-TI-User. Verkaufe org. Ex Basic VB 165 DM. Angebote an Lukas Merten, Marktstr. I9, 5440 Mayen

---

Suche Original E/A Handbuch. Reimund Müller, Buchenstr. 20, 4630 Bochum 6, Tel. 02327/83764

Suche Modulexpander u. ausführliche Bedienungsanleit. für ext. 32K-Speichererwe. mit Centronic-Schnittstelle (auch Kopie) Angebote an: Berth. Fella Hardenbergstr. 28, 8500 Nbg 20, Tel. 0911/549804 (rufe zurück!)

Verk. für TI/99 4A BASF Laufwerk 6106 VB 300 DM Tel. 0208/73042

Verk. TI99/4A + Ext. Basic Modul, Preis VB. Tel.: 02151/405869

Verk. TI99/4A incl. Handbuch für DM I20,—. Alexander Rupp, Kalmanstr. 45, 6600 Saarbrücken. Tel.: 0681/45134

Verkaufe TI-99 100 DM
Exd. Basic + d. Handbuch 150 DM
Parsec (Mod.) 30 DM
TI-Schach (Mod.) 50 DM
Blasto (Mod.) 8 DM
Orig. TI-Recorder + Kabel 60 DM
Joysticks 10 DM
Thomas Schmidt, Eiswiese 7, 5160 Düren
Tel. 02421/42488

TI99/4A (def) + Orig. XBasic + deut. Handbuch + Rec. Kabel + Literatur. Preiswert! G. Schulte, 0421/71149 ab 17 h

---

Verk. TI-Konsole + XBasic + 1MB-Interface + 64K-Dram-Karte + Zenith-Monitor VB 950 TI59+PC100C VB300 Tel. 0941/67884

Suche Peribox f. TI99/ 4A Diskettenlaufwerk 32K Erw. Artur Kasimir, Weslarnerweg Nr. 7, 4770 Soest, Tel. 02921/8591 nach 18 h

---

Drucke Listings kosten nur Rückporto beilegen. Verkaufe Spkel 1+2, Basic Inf. Marketing Plan Spiel 10,– DM

Suche ext. Laufwerk + Disc-Kontroller günstig. Telefon 06805/8393

TI 99/4A * Verk. Super Sketch (Grafiktablett) Markus Schenk. Tel. 07136/ 4116 ab 17 h

---

!!!Extended Basic!!! Verkaufe TI99/4 + Extended Basic + Handbuch + Recorder-Kabel (340,— DM) 0711/ 537789

Extended Basic Modul 180,—
Disc Manager II Modul 110,—
Parsec 30,—
Siegfried Fränkl, 0941/96296 ab 16.30 tägl.

Wer hilft 99/4A-Fan i.d. DDR? Gesucht: billiges 32K RAM, Konsole z. ausschlachten, Schaltplan und Bauanltg. f. Hardware, def. Hardware jed. Art. Kontaktadresse: Otto 42 OB 1, Feldmannstr. 54, Tel. (ab 19 h) 0208/860892

ANDREAS EHLERDING TECHNOLOGIE

Ein neuer Name, wenn es um Computer geht.

Er steht für Forschung und Innovation an der Nahtstelle von Psychologie, Medizin und Informatik. Trotz finanzieller Engpässe konnten wir bei unseren Computern von vornherein nicht auf kompromißlose Zuverlässigkeit und höchste Flexibilität verzichten. So fiel die Wahl auf den TI 99/4A. Sehen konnte der TI bisher noch nicht, doch dank unserem VIDEO DIGITIZER AET - VD 99 ist dieses Handicap nun beseitigt.

DER BLICK IN DEN SPIEGEL

TI 99/4A mit angeschlossenem Video Digizer AET;VD 99.
Vergrößerter Bildausschnitt, 64 x 48 Punkte,
16 Farb bzw. Stufen mit spezieller Software in druckbare
2-Farben Darstellung umgewandelt.



WANN WIRD IHN IHR TI TUN?

Technische Daten:

64000 Bildpunkte 64 Grauwertstufen · belegt keinen Speicherplatz im TI · 64 K eigenes RAM, auch als Erweiterung nutzbar 32 K CMOS-RAM (auf Wunsch zusätzlich integriert) · incl. Bildverarbeitungs-Software · Darstellung der Bilder auf dem Bildschirm 256 x 192 Punkte · Objekterkennung (lernen durch zeigen) · Grauwerthistogram · Kantendetektion · Ausschnittvergrößerung eines beliebigen 64 x 48 Punktebereiches in Hardcopyroutine für Epson FX 80/RX 80

1498,– DM incl. MWSt.

ANDREAS EHLERDING TECHNOLOGIE
BERATUNG ENTWICKLUNG FERTIGUNG SERVICE · NIEDERSACHSENRING 26 · D-3051 WÖLPINGHAUSEN 05037/744

Achtung!!!

RADIX

RADIX Bürotechnik
Rappstraße 13 · 2000 Hamburg 13
Tel. 040/4416 95 · Telex 213 682 radix d
tägl. 10.00-12.30 + 13.30-18.30 Uhr
Sa. 10.00-13.00 Uhr
Verkaufsstelle Kiel: Ziegelteich 23 · 2300 Kiel 1

IHR TI-SPEZIALIST

hält für Sie bereit:

32 KB Erweiterung
extern Batterie gepuffert **258,–**

GPL-DISASSEMBLER

auf Diskette
notwendig 32 K und
Editor Assembler
zum Auflisten von 6 ROM-Module
59,–

Preisliste bitte anfordern

IMMER NEU UND AKTUELL FÜR TI 99/4A

EXTENDED-BASIC (Mechatronic)
mit deutschem Handbuch 199.90
EXTENDED-BASIC II PLUS mit deutschem Handbuch 299.—
= Extended-Besic + Grafik Extended-Basic (Apesoft) in 1 Modul

Umtauschaktion
Bei Bestellung eines EXTENDED-BASIC II PLUS vergüten wir Ihnen DM 70.— bei kostenfreier Zusendung eines orlginel amerikanischen Extended-Basic-Moduls (elektrisch/mechenlsch einwendfreier Zustand!!)
Sie zahlen nur noch 229.—

Umbeuaktion (gilt nur für deutschen Lizenznachbau „Mechatronic"). Wir machen aus Ihrem EXTENDED-BASIC ein EXTENDED-BASIC II PLUS mit deutschem Handbuch für nur 98.—
32-k-RAM-ERWEITERUNG mit Centronic-Interface, Kunststoffgehäuse 190 × 110 × 60 mm zum seitlichen Anstecken en den Bus, der Bus wird nach rechts durchgeschleift, mit 5-V-Steckernetzteil 289.50*

Unser Paketpreis-Angebot
EXTENDED-BASIC II PLUS + 32-k-RAM-ERWEITERUNG, ohne Centronic-Interface für nur 499.50*

NEU! 128-k-RAM-ERWEITERUNG, mit Centronic-Interface und 5-V-Steckernetzteil 595.—

NEU **Die Weltneuheit: 128 kB — GRAM Preis ca. 750,—**
Lieferbar etwa Janur 1986

Preise in DM/Stück Inkl. MwSt. · Technische Änderungen vorbehalten
Versand gegen Nachnahme oder Voreuskasse.

4-FARBEN-PRINTER-PLOTTER PP-A 4, Centronic-Schnittstelle, DIN-A 4-Format, Direktanschluß en 32-k- oder 128-k-RAM-Erwelterung 699.—
ANSCHLUSSKABEL
von 32-k- oder 128-k-RAM an PP-A 4 68.—
SLIM-LINE-LAUFWERK 5,25", 500-k-Byte-DS/DD
(z. B. TEAC FT 55 B) 399,90—
EINBAUSATZ für 2 Laufwerke in originel TI-P-Box 95.—
DISC-STEUERKARTE (CorComp), DS/DD, für max.
4 Laufwerke 635.—

NEU! OUICK-DISC-FLOPPY (im Gehäuse), zum Direktenschluß an die Konsole, **keine Steuerkarte erforderlich,** 128-k-Byte-DS, für 2,8"-Disketten, mit 5-V-Steckernetzteil, identisch mit der bekannten MSX-Version 598.—

SEHR NEU
DISKETTEN 2,8", 10er-Pack 95.—
TI-MAUS — die schnelle und komfortable Cursorsteuerung mit Softwere euf 5,25"-Diskette, mit 5-V-Steckernetztell 296.—

NEU VIERSPANNUNGS-SCHALTNETZTEIL, +5 V, 4 A/±12 V, 0,3 A/ –24 V, 0,3 A, primër getaktet, 35 Wett, MOS-Fet-Technik, extrem klein (80×125×32 mm), offene Bauweise, ideel zum Betrieb von Druckern, Monitoren etc. 345.—

*** Preissenkung — denk großer Nechfrege!**

albs-Alltronic G. Schmidt · Postfach 1130 · 7136 Ötisheim
Tel. 0 70 41 / 27 47 · Telex 7 263 738 albs